A FILING SYSTEM AND SPOOLER FOR THE TDC-316 TIME-SHARING SYSTEM

A Thesis Submitted
In Partial Fulfilment of the Requirements
for the Degree of
MASTER OF TECHNOLOGY

by

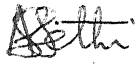
AARTI KUMAR

to the
COMPUTER SCIENCE PROGRAMME
INDIAN INSTITUTE OF TECHNOLOGY, KANPUR
JULY, 1982

Dedicated
To My Brother
R A K E S H

CERTIFICATE

CERTIFIED that the work entitled "A FILING SYSTEM AND SPOOLER FOR THE TDC-316 TIME-SHARING SYSTEM" by Aarti Kumar has been carried out under my supervision and has not been submitted elsewhere for a degree.



A.S. Sethi
Assistant Professor
Computer Science Programme
INDIAN INSTITUTE OF TECHNOLOGY, KANPUR

Kanpur
July 1982

ABSTRACT

A filing system and a line printer spooler have been designed for TDC-316. Both of them run in the environment of a time sharing system which allows 16 on-line users. Editing of files is envisaged to be the most important activity in the user environment of this system. The filing system is interfaced with the operating system in the shape of twelve primitives. These collectively perform all the filing functions. The filing model has a hierarchical structure which is generalized to allow for its portability to any other operating system designed for TDC-316. The spooler runs as a user process. Other users communicate with it to get their files printed.

ACKNOWLEDGEMENT

I thank Dr. A.S. Sethi, my thesis supervisor for his guidance and encouragement during the course of this work. The regular discussions with him have been very fruitful in completing this project.

I thank Sri S.L. Agarwal and Sri A.K. Basu for their able maintenance of TDC-316 and for the everready help rendered to us.

I thank my project partner Mr. P.K. Pandya for his team spirit and his bright ideas.

I thank George Paul for his suggestions during the designing and invaluable help during the debugging process.

My brother Rakesh Kumar and Mr. Sunil Jha have been a constant source of encouragement. I thank them specially for their much needed assistance in the last few days.

I thank Mr. Mohammed Anwar for his neat and patient typing.

I thank all my friends in the girls hostel and my classmates for their companionship.

Finally I thank all my teachers here for their guidance and help during the course of my M.Tech.

KANPUR
JULY 1982

- AARTI KUMAR

CONTENTS

CHAPTER

CHAPTER 1

INTRODUCTION

This thesis along with a companion thesis [PAND 82], is concerned with the design and implementation of a small time sharing operating system for the TDC-316. This is a mini-computer of 16 bit wordlength built by the E.C.I.L. Ltd. on the lines of the DEC PDP/11.

Each operating system is designed to serve some specific purpose. Our design was motivated by the needs of a group of users who are new to computer systems and inexperienced programmers and terminal users.

Such a group is best represented by a number of students in an educational institution doing a first course in programming. Their computational efforts are often localized to the editing and running of small programs with a view to learning both the language and terminal usage.

Since our purpose has been to provide such users with adequate facilities, the system as it stands now offers time sharing facilities to a small number of users. (A maximum of 16 on-line users are possible). It also provides users with adequate filing facilities and an extremely fast line oriented test editor. There is also a card reader and line-printer

spooler, allowing the creating of files through cards and the printing of files.

A significant factor in the design is the absence of any language support and hence the facility to execute programs on the machine. This was omitted mainly due to lack of memory in the TDC-316. An alternate provision was made by linking the TDC-316 to the DEC-10 via asynchronous communication line.

Now users can create files on the TDC-316 and submit their programs to the DEC-10 (through the TDC-316) for execution. Since the DEC-10 supports several languages, the programs can be easily executed and the output returned to the TDC-316 through the DEC-TDC link. The output file can be printed by the user on the TDC-316.

Thus the design of the operating system was influenced by the choice of the machine and its inadequacies, the users expected and the linking of the TDC-316 to the DEC-10 which allowed us to omit the execution of programs on the TDC-316 itself.

The design of the filing system was influenced by another factor. It was felt that the filing system should, if it ever be so desired, be transferable to any other operating system written for the TDC-316. Hence its structure has been generalized and its interface with the monitor has been kept relatively simple. The system has a tree structured directory scheme and a protection mechanism which distinguishes between

3 classes of users (a) owner (b) of the same project and (c) others. The interface with the monitor is in the shape of twelve primitives which collectively perform all the filing functions of creating, destroying and modifying files.

1.1 DESIGN METHODOLOGY AND IMPLEMENTATION

The operating system has been designed and implemented in a relatively short time space. Many factors contribute to this. One was the choice of the language BLISS-11 for implementation.

This language was chosen since it was the only system software language available on DEC-10 for use with the TDC-316. The BLISS compiler translated the BLISS code into PDP-11 code. Also available was a cross assembler MACN16 to convert the PDP-11 code to the TDC-316 object code on the DEC-10. This generation of TDC-316 code on the DEC-10 cut down on implementation time. A loader to directly load object programs from the DEC-10 to the TDC-316 via asynchronous lines was also available and hence programs developed on the DEC-10 were directly loaded onto the TDC-316, from where they could be executed directly. There was a linker available to link all programs together and run the combined object code version on the TDC-316.

The design of the operating system was kept relatively simple and carried out in a top-down manner, initially leading from general specifications on what the system is expected to

do to details on how to perform the functions. At the lowest level in the design is the kernel which is the heart of the system and synchronizes and manages all processes and resources.

1.2 OUTLINE OF THE THESIS

The work on the design and the implementation was divided into two areas, one which dealt with the development of the kernel and the management of all resources and the other which dealt with the design of the filing system and the line printer spooler. A third area is the development of the editor which was taken up separately.

In this thesis, the details of the filing system and the lineprinter spooler has been given. The second and the third chapters deal with the hardware and the software environment of the operating system. The fourth and fifth chapter give the details of the filing system and the disk management. The spooler is described in the sixth chapter. The seventh chapter gives some idea of the accounting and error handling. The same chapter elaborates on improvements possible in the design. The appendix contains the program listings.

CHAPTER 2

HARDWARE ENVIRONMENT

In this chapter a brief outline of the mini-computer TDC-316 has been given. First certain features of the hardware have been described. Later a section on the disk drive controller has been included. The last section gives a general description of the line printer controller.

2.1 THE TDC-316 COMPUTER

The TDC-316, the machine for which this operating system has been designed is a small, high speed mini-computer with a word length of 16 bits and parallel 2's complement arithmetic [TDC].

The basic systems has 28K words of core memory expandable to 128K words with the MAP option. It has a dual bus structure - the intermixed bus and the memory bus. High speed devices like the magnetic tapes and disks can be connected as DMA devices to either bus. There is an 8 level automatic priority interrupt scheme provided with the TDC-316. DMA devices can gain control of the bus by generating non-processor request signals which have higher priority than interrupts. Nesting of interrupts is allowed.

The system has 15 addressable general purpose registers. Besides these, it has a few other special registers like the processor status word. This at any time gives the current state of the machine. The processor can run in 2 modes, the user (or unprivileged mode) and the supervisor (or privileged mode). It maintains two stacks, one for each mode of the processor. There are powerful trap features built in the system to detect illegal program specification and certain hardware conditions.

The MAP option maps out different physical spaces in memory in the user and supervisor mode. The MAP option is used to provide the mapping or relocation of the logical address space available to the programmer into a larger physical address space. In each mode 36K of virtual memory is available (28K for Instruction and 28K for Data). In addition 4K of I/O cum ROM addresses is common to all spaces.

In each area, the 32K total addressable space is further divided into 8 segments each of whose length can vary from 32 words to 4K words. Each segment must begin from a 32 word block boundary. The MAP option has hardware registers to keep the starting addresses of each segment in each mode.

The TDC-316 has all the standard peripherals of a line printer, card reader, high speed paper tape and punch and a disk unit. In the subsequent sections, we describe both the line printer and the disk unit in some detail.

2.2 THE DISK DRIVE CONTROLLER

The controller has a central section termed the adaptor and a section to control the disk drive peculiarities. More than one disk drive can be connected to the central section. Basically the adaptors functions are to receive, interpret and rout commands to the drive for which they are meant. It also controls data transfer by assembling and disassembling the data flow, providing intermediate buffers etc. On the completion of a job it transmits an interrupt signal to the CPU. [TDC].

The formatter takes care of each drives peculiarities. It executes the commands passed on by the adaptor and provides the interfacing between the drive and the controller. The disk subsystem has 2 interrupt vectors, one for completion of data transfer operation and the second for non data transfer operations. These vectors are located at addresses #200 and #204 respectively.

2.2.1 The Disk Unit

We use the disk pack drive at our installation. It has an unformalted capacity of 7.25M bytes. There are 203 tracks, 10 sections per track and 10 surfaces.

There are eight registers maintained here.

1) Sector and Surface Register (Address #177444) : This stores both the sector and surface number.

2) Attention Register (#177446) : An interrupt on account of a non data transfer operation sets a bit here which specifies the port interrupting. This is necessary since more than one drive can be connected.

3) Drive Status Register (#177450) : This gives the status of the drive. It can be read after a SENSE is executed. (SENSE - refer to Command Register).

4) Track Register (#177452) : This stores the current track number.

5) Command Register (#177454) : Through this various commands can be given to the disk controller. There is a SENSE command, which when set loads the current port and unit number in the DRIVE STATUS Register, which can now be read.

6) Controller Status Register (#177456) : This is a read only register. It is kept in the adaptor and contains status information regarding mainly data transfer operations.

7) Current Address Register (#177460) : This keeps the memory address from/to which the data transfer must take place.

8) Word Count Register (#177462): In this must be loaded the 2's complement of the number of words to be transferred.

2.3 LINE PRINTER CONTROLLER

The line printer provided with the TDC-316 prints 132 columns across the line at the rate of 300 lpm using a full line buffer. Characters to be printed are given to the printer serially and are stored in the line buffer of the printer. [TDC]. Four registers are maintained here -

1) Line Printer Control and Status Register (LCSR) #777630
2) Data Buffer Register (LPBR) #777632
3) VFU Register (LVFR) #777634
4) Current Count and Status Register (CCSR) #777636

The current count register provides the line count and the character count. Line count is cleared after a page eject command. Paper movement can also be controlled by the VFU register. Characters are loaded into the printer buffer until the 'Buffer full' bit is set. Now a print command may be given through the LCSR. The characters are loaded in 6-BIT ASC11 code. All lower case characters are converted to upper case and printed. Two interrupt vectors (1) error interrupt (#160) (2) ready inteerupt (#164) are available.

CHAPTER 3

SOFTWARE ENVIRONMENT

The design of this operating system has been kept relatively simple. Here to each terminal is attached a job which automatically serves as the users job when he logs in. There are no back-ground jobs. [Thom 72].

In the main memory, space is allocated for the kernel programs the editor and other utilities. The rest of the core space is divided into users' work spaces, depending on the number of users, a parameter that can be specified when the system comes up. Each user has a communications area, a data area and a stack area. In the user mode of the process the MAP OPTION is set up to point to the current users work area, otherwise in the supervisor mode the MAP points to the kernel routines.

An interrupt from any peripheral device or the clock and the invocation of any trap call results in a switching from user to supervisor mode. Here the supervisor map is used. Still both modes share a common instruction segment, which is segment No.6 in the supervisor mode and segment 0 in the user mode.

The filing routines all run as an extension of the user's job but since this code is sharable, all users must have access to these routines. Therefore for each user there must be a common segment available containing the filing routines. A user can invoke the filing primitive routines via trap calls. The user's map is then changed to point to the appropriate routines and the user's PC is loaded with the start address of the invoked routine. A user in this state is then said to be in the FILING MODE. This is the 3rd mode of the processor but it is entirely simulated by software. It is useful in the sense that any disk I/O request by the user can only be made in this mode since the disk driver is hidden from the user in the filing system. Hence a security check is done at the time a request for data transfer from the disk is made by any user.

Certain data segments in the filing system are locked when being used by one job to preserve the integrity of the system. These can be thought of as resources and since all users can invoke the filing system concurrently, they must compete for the use of such resources. With each resource is associated an event number. Binary semaphones and event queues are used to synchronize the allocation and release of these resources. A user may thus be blocked on a particular event and put in the appropriate queue. When a running job releases the resource, the first job in the event queue associated with that resource is woken up [BRIN 73].

Now each invocation of a filing primitive may result in a sequence of disk I/O operations. Since all the primitives are run as an extension of the user's job, the job must be blocked at the initiation of any disk I/O and woken up when the DMA transfer has taken place. Thus here too a blocking and awakening sequence is used but there are no queues maintained.

The blocking, awakening and queue management routines are a part of the lowest layer of the kernel. For further details a companion thesis [PANDA 82] may be seen.

When a user's job is blocked on initiation of disk I/O, the next user to invoke a filing primitive may need to do some data transfer from/to the disk too. Since a new I/O operation cannot be initiated till the first is completed, the user's request for disk I/O must be entered in a queue and taken up later. This queue of requests made by jobs to accomplish a disk I/O operations is maintained by the disk manager. A user's job is blocked when his request is entered in the queue. The interrupt servicing routine of the disk wakes up the last job for whom the I/O is completed and takes up the next request in the queue. This queue is thus interrupt driven.

Since all the interrupt servicing routines run in supervisor mode, the queue and its management have been made a part of the kernel which too runs in supervisor mode. Now the user enters his request via trap calls.

Here since all data transfers are to or from memory locations in user's area, we need inter-mode communication. Thus either along with a request the 18 bit actual physical memory address is loaded, or at the time of initiation of I/O a routine is called to convert the relocatable 16 bit address into an 18 bit address in user's area.

Both the card reader and the line printer spoolers on the other hand are treated as user processes in themselves. All other users communicate with them to get their files read or printed. The spooler too maintains a table of print requests. This table is accessible to all users for entry of their requests. All requests must be regarding files stored on the disk. The spooler being a user's process can invoke filing calls like OPEN and GET. Also at the initiation of any disk I/O operation it will get blocked. Like all users, it maintains a single file buffer in its area as long as file is open. The main purpose of the spooler is to get sequential blocks of the file and transfer characters line by line to a ring of buffers in the kernel. This ring of buffers then contain lines which are printed in the order in which they are filled.

Since the operating system has been designed basically to cater to the editing needs of users and to provide them adequate filing facilities and as all the users expected to use the system have a common purpose, no user's job is given priority over another. Hence the jobs are scheduled in a

round-robin manner. Also there is no swapping in the memory management. A buffer of the size of one disk sector is kept fixed for each open file in the user's area and is used for all data transfers concerning the file. Since text editing is the main purpose of the system, a file is created only as a stream of characters. A file may also be created through cards but only by a logged-in user.

Error handling is of two kinds. One due to which jobs are aborted and the other where only appropriate messages are relayed back to the user but the job continues to be in run state.

A link has been established with DEC-10 which supports several languages so that users may submit their files created on the TDC-316 for execution on the DEC-10.

CHAPTER 4

FILING SYSTEM

We first give a brief picture of the overall system as a hierarchical model. The detailed description of the design of each layer in the hierarchy is then specified. This is spread over Chapters 4 and 5.

This chapter deals with the concepts and details of the filing model which converts a user's request for some filing function to a sequence of actual disk I/O requests. The peculiarities of the disk driver and the scheduling and control of disk I/O requests is the subject matter of the 5th chapter.

4.1 HIERARCHICAL MODEL

A general description of this model has been given without making each layer specific. The attempt has been to describe the system from the outermost layer, namely that which is visible to the user and the innermost layer of actual disk I/O operations [SHAW 74].

At the outset, we shall look at the filing system from a user's view. For a user, filing facilities are available either directly by invoking monitor commands or indirectly through utilities like the editor, spooler etc.

A directory of files owned by each user is maintained. This directory may contain text files and other directories and any level of such nestings of directories may be allowed. Since the directory structure is a tree, each user's directory has a path in the tree and a user can access his files by giving a path specification. The path can be specified from the user's current directory (into which he logs-in) or from the root directory.

Each file has a descriptor which contains all relevant information about the file. Descriptors also store the disk sector addresses of the blocks allocated to the file. A file may have more than one descriptor and hence can be of any length since all its descriptors are chained together and give sequentially the addresses of the file blocks. There is a generalized protection scheme dividing users of files into (1) owner, (2) of the same project and (3) others and is used effectively to control access to files.

If a user wishes to read or write into a file, the file must be activated. This implies that the file descriptor must be read into the memory. A table of all active files is maintained. This table is searched to ensure READ-ONLY sharing of files when a file is opened (i.e. activated). Thus if a file is opened for READ-WRITE, there must be only one active copy of the file.

File blocks may be accessed by using logical block numbers starting from 0 which points to the first block.

All data transfers regarding the file take place from or to a fixed buffer in the user's memory; specified by the user when the file is opened. Thus the filing system does not itself do any buffer management. A file is always treated as a stream of characters and all transfers are in terms of file blocks of one sector size. There is no notion of records. This can be built over the system easily if required since random accessing of blocks is possible.

When the user has completed all transfer operations on a file, the file is closed i.e. deactivated and the entry in the active table is freed.

Free blocks on the disk are those which have not yet been allocated to any file. A linked list of all such blocks is maintained, and allocation/deallocation of blocks take place from the head of the list. All descriptors are kept on certain specified tracks. Their current state is maintained by a BIT MAP spread over 2 blocks in track 0. A bit when set indicates that the descriptor pointed to logically is allocated to a file, when reset it implies the descriptor is free.

Since filing routines can be invoked concurrently by all users, a queue of requests to facilitate the actual data transfer operations is maintained. This queue is interrupt driven, jobs are blocked on entering their requests in the queue and woken up after the I/O is over. This

section of the filing system i.e. the scheduling and controlling of all I/O requests is maintained in the kernel. There is a header block which keeps all the information about the disk organization; including the head of the free block chain, the addresses of the blocks containing the BIT map for the descriptors and other such specific information. This is always kept in memory when the system is initialized and written back when any modification is made, while still keeping the new copy in memory.

These in a nutshell were some of the salient features of the filing system. Detailed descriptions are given in the following sections. The appendix contains description of all relevant tables.

4.2 DETAILED DESCRIPTION OF THE SYSTEM

A file system is that software responsible for creating, destroying, organizing, reading, writing, modifying and controlling access to files and for the management of resources used by files. The filing facilities are available to the user by invoking any one of the 12 primitives:-

CREATOR, CREDIR, OPEN, CLOSE, DELETE, RENAME, PUT
GET, PUTEOF, BLKCHK, CDPCHK, SEEK (See Appendix I)

4.2.1 Directory Structure

The directory structure of the system is a tree and each directory is stored at some node in the tree. Directory

entries are branches which point to other directories or to text files which form the leaves of this tree. At the root of the tree is a master file directory owned by the system. Here nesting of directories to any level is possible hence a user may have many levels of SFD's in his area. The system deals with only 2 kinds of files - directory and text. Each file has a main descriptor which contains relevant information about the file [RITC 78a] . Figure 4.1 depicts an example of such a directory structure.

4.2.2 User's View

A file, as mentioned earlier, is uniquely identified by its path in the directory tree structure. The user can specify the path as follows:-

Directory 1/Directory 2/Directory 3/...../Filename

Here search for the file begins from the users directory pointed to by its CDP (i.e. current directory pointer). To specify the path from the root, there must be a leading "/" (slash) in the pathname. This description is at the monitor level [RITC 78b] .

To the filing system, a pointer to the pathname is passed, which itself is stored in contiguous words in the communications area of each user by the system. The format here must be as given below:-

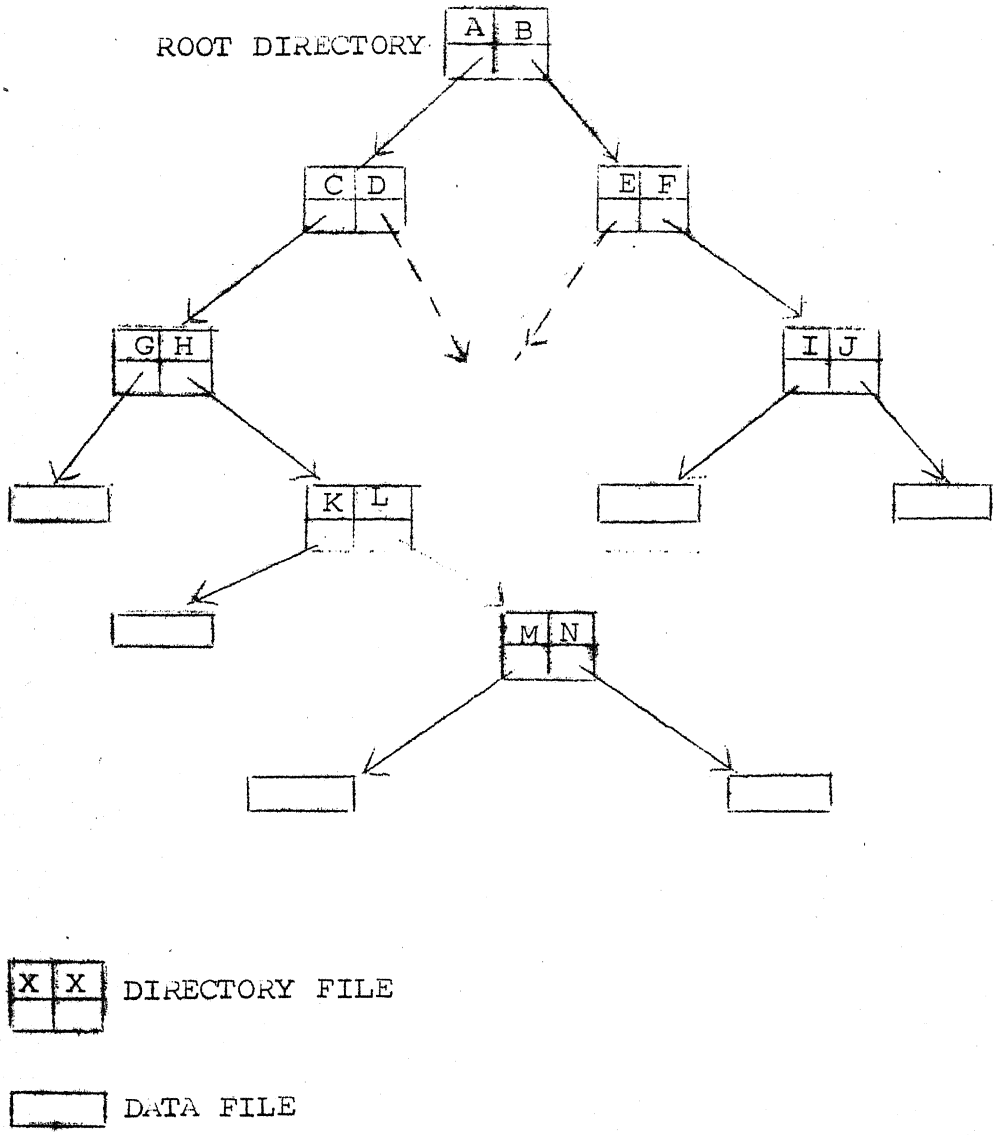


FIG. 4.1 A TREE STRUCTURED DIRECTORY

[1st word] This contains an address as given by the CDP or -1 if search begins from the root.

[FILE NAME] [FILE NAME] [FILE NAME].... [FILE NAME] [EOLIST MARK]

Each filename is 2 words long and all filenames except the last must refer to a directory file on the path. The EOLIST MARK is a '$' (dollar) sign placed at the end of the list.

4.2.3 Descriptors

The descriptor for a file is 64 words long (half a disk sector size). Its contents are typically:-

1) Filename in radix-50 code.

2) Owner's PPN.

3) Kind of file (i.e. directory or text).

4) Protection for the file.

5) Address of the directory file to which this file belongs.

6) Its own address

7) Address of the next descriptor of the file (or -1 if the file has no more descriptors).

8) Date of creation.

9) Storage map.

10) Size of file (i.e. number of disk blocks allocated to the file).

In the descriptor there is space for storing 54 addresses of blocks allocated to the file. If the size of a file is more than 54 blocks then another descriptor is allocated to the file and links between the 2 descriptors are maintained. This chaining of descriptors for a file is possible to any number implying thereby that there is no restriction on the size of the file [RITC 78b].

Second and later descriptor contents are (1) address of the previous descriptor (2) address of the current descriptor (3) address of the next descriptor if any or else -1 and (4) storage map for the file.

4.2.4 CDPCHK & CREDIR

To set the current directory pointer for a user, he must specify the path of the directory. The current directory pointer or (as we shall call it from now on the CDP) essentially gives the address of descriptor of the directory file to which a user logs into.

The primitive CDPCHK is invoked for this purpose and the path of the directory from the root is traced and checked for read-only access atleast for the user. CREDIR is a primitive invoked to create a directory file. The user/system wishing to create a directory must specify its owner alongwith the primitive. But for text files the owner is the same as the owner of the directory to which the file belongs.

A directory file is treated differently from a text file

in that it cannot be explicitly written into. Entries are made in the file when a file is created or entries can be removed when a file is deleted. Reading of a directory file implies the looking up of filenames of files belonging to the directory. Figure 4.2 shows a typical block in a directory file.

Each entry in a directory is 3 words long. First two contain the filename of the file and the 3rd word contains the descriptor address of the file. After the last entry in the file a mark indicating the end of file is put. When a file is deleted, its entry in the directory is replaced by a -1 sign in the first word.

The root directory is stored at a fixed block and initialized when the system is set up. The address of the root directory is stored in the Header Block of the file.

4.2.5 Protection Mechanism

Users of files can be divided into 3 groups (1) owner (2) of the same project and (3) others. A hierarchical scheme is used to control the access. Each group of users are allowed either one of the following [MADN 74] .

*) No access Value 4

*) Read only access Value 3

*) Read write access Value 2

*) Read, write, delete access Value 1

*) Read, write, delete & change protection Value 0

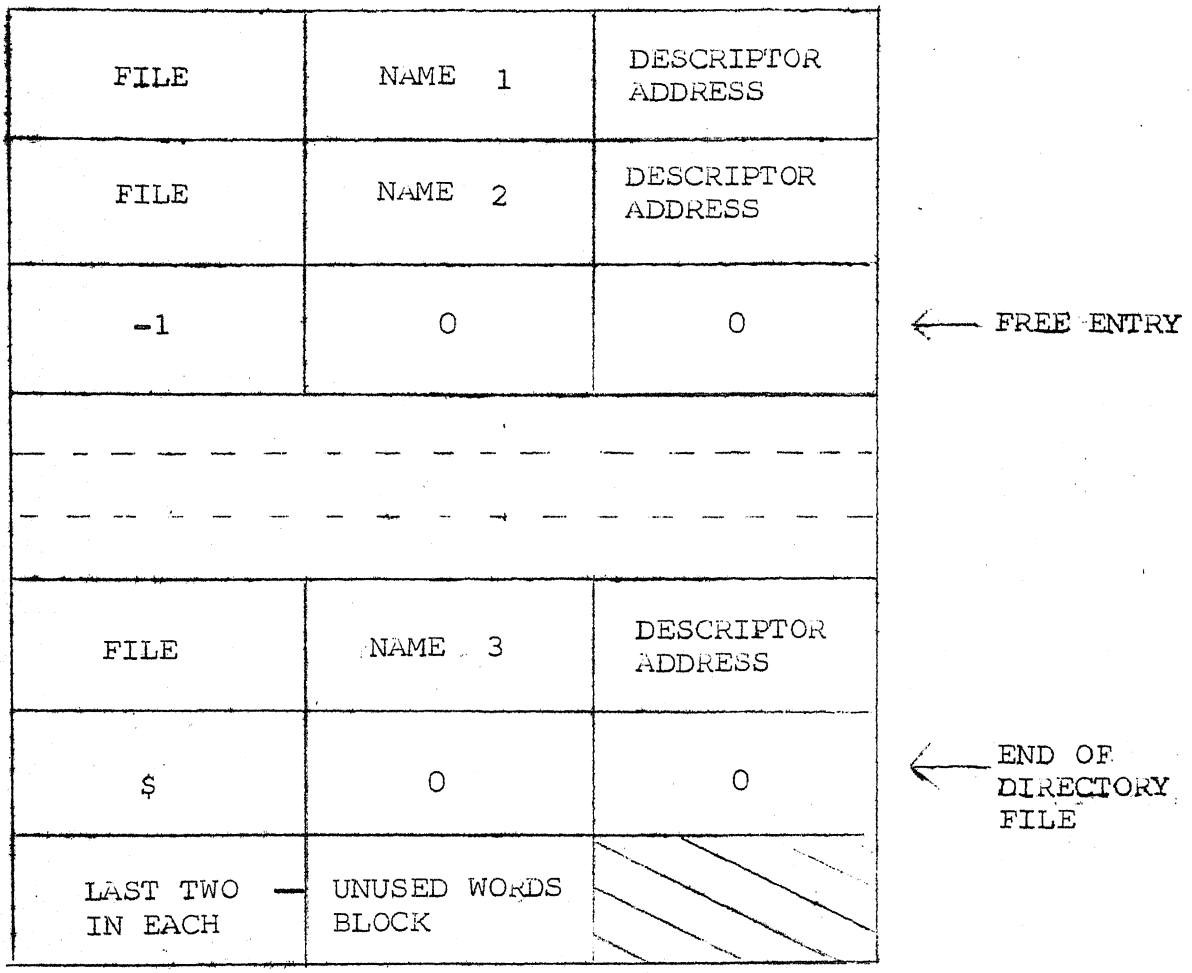


FIG. 4.2 A TYPICAL DIRECTORY BLOCK

Here a 3 digit number with each digit in the range 0 to 4 is used to keep a check on the kind of access made by any user on the file. When a user wishes to access a file, he must be allowed to look up directories on the path of the file from the root. Hence he must have atleast read only access on all directories on the path. If he wishes to create, delete or rename a file then the user must have write access on the directory to which the file belongs, and he must have the specified access on the file itself. The only variation in this rule is in the case of owners who are always allowed to change their protection whatever be the protection specified earlier.

Thus a general protection mechanism is used to control access to all files. This also imposes an order in which directories are allocated to users. Since users must have read access on all directories on the path, hence the path must have directories owned by users who are willing to let other users in the path look into their area. This is general, in the sense that a user may create a file in another users directory provided he has write access on the directory and read access on the path; but for deleting another user's file, the first user must also have delete access on the file.

4.2.6 Activation of Files

When a user wishes to access a file for reading, writing or modifying the file must first be activated. That

is, the main descriptor of the file must be read into the memory. To facilitate this a table of all active files is maintained. Each entry contains a pointer to the descriptor of the file in memory. Such a table is called the ACTIVE TABLE (See Appendix II for a detailed description of this table).

A file is activated by invoking any one of the two calls, CREATOR and OPEN. CREATOR creats the file entry in the directory, allocates and initializes a descriptor for the file and then activates the file exactly like in OPEN. This primitive is used to create only text files [SHAW 74, SHAW 79].

First the block address of the descriptor of the file is obtained (both in CREATOR and OPEN) after checking for atleast read only access on all directories on the path and in the case of CREATOR, for write access on last directory. Hence the user's PPN must be supplied with the primitive.

A file may be opened in either one of the two states, READ and READ-WRITE. Since read-only sharing of files is allowed, before a file is opened the ACTIVE TABLE is searched and the descriptor address of the file is matched since this address uniquely identifies a file. If the file is to be opened for READ-WRITE and it has already been opened earlier, an error message is returned. Again if a file is to be opened for READ and it has been opened for READ-WRITE earlier, the error message is returned and the job is aborted in both cases.

A file to be created is always opened for READ-WRITE and is of size 0. Whenever a new block is allocated to the file, the size is increased by one.

To open a file for READ-WRITE the user must have write access for the file and to open for READ the user should have read access in the file.

With each file certain other status information like the user's job no., the current size of the file, the current descriptor number for each file is maintained. Other pointers and flags like the current block pointer etc. too are kept. Their use shall be clear from the next section.

Channel numbers starting from zero are used to index the ACTIVE table. These are also returned to the user invoking CREATOR or OPEN. Now an active file can be accessed by the job which opened it by just referring to the channel no.

Directory files can never be opened for writing and hence CREDIR does not return any channel number to the user.

4.2.7 Reading, Writing and Modifying

For each open file in the Active Table, is maintained a current block pointer. All reads and writes of the file are with reference to this pointer. A file is treated only as a stream of characters and users may get (i.e. read) blocks of the file into their memory area and also put (i.e. write) blocks of the file from their memory area. The current block number is initially zero and points to the first address

in the storage map of the file descriptor. Thus specific blocks of the file can be accessed by their logical block numbers from the start of the file. When a GET or a PUT primitive is invoked, the logical block number is obtained from the current block pointer. The address of the specified block can now be read from the descriptor. This address is used to initiate the data transfer. After the operation is over, the current pointer in incremented and the new value is returned to the user. The next GET or PUT will result in a data transfer from the block currently pointed.

A user can set his pointer to any specific block by using a SEEK command. A user may also check the current value of the pointer by a BLKCHK command [RITC 78a].

Another primitive available here for ACTIVE files is the PUTEOF. This puts an end of file mark in place of the address pointed to by the current block pointer in the descriptor. The remaining blocks allocated after this mark are then released.

The pascal primitives RESET and REWRITE can be implemented easily. For RESET a SEEK (0) will reset the pointer to the beginning of the file and for REWRITE a SEEK (0) and a PUTEOF after it will delete all the blocks of the file and reset the pointer to the beginning, while still keeping the file ACTIVE.

For all data transfers regarding ACTIVE files the user

must specify a memory buffer in his area, with the OPEN or CREATOR primitive. The address of this buffer is stored in the ACTIVETABLE entry for his file.

4.2.8 Deactivation of Files

A file may be deactivated, essentially made inaccessible to the user for reading or writing purposes by a CLOSE command. This primitive frees the ACTIVE TABLE entry for the file SHAW 79,[RITC 78a].

When a user changes the size of the file, i.e. either by specifying a PUTEOF which frees some file blocks or by specifying PUTS which result in the allocation of a new block, a modification must be made in the size of file field in the first descriptor. This is done when a file is CLOSED.

The current size of the file is always maintained in the ACTIVE TABLE. A bit to indicate if the size has changed from its original value, is also kept. If this bit is set, the first descriptor of the file is read into the memory, the size of the file is modified in the descriptor and then it is written back.

Another flag to indicate if the current descriptor is modified is also stored. This is to ensure the writing back of the current descriptor (if modified) before another descriptor is read.

Each descriptor of a file is linked to the preceding and succeeding descriptors. Also the address of the first descriptor is kept in the ACTIVE TABLE entry for the file.

4.2.9 DELETE & RENAME

These are two more primitives whose use so far has not been mentioned. A user may invoke DELETE if he wishes to remove a file from some directory. To delete a file, the user must have read access on all directories on the path from the root, write access on the last directory and delete access on the file itself.

A directory file can only be deleted when empty. If a file is ACTIVE when a DELETE primitive for the file is invoked, then the directory entry of the file is removed while still keeping the blocks of the file. The file is now in the DELETE state. The last CLOSE command for the file then frees the blocks and descriptors of the file. No other user can OPEN the file after the entry is deleted from the directory.

The RENAME primitive is invoked if a user wishes to change the name or the protection rights of the file. In both cases the user should have write access on the directory of the file. For changing only the name, the user must have "delete" access on the file itself and for changing the protection the user should have "change protection"

access on the file [DEC] . With this section we conclude the description of all the primitives.

4.2.10 Free Storage Management

In the preceding sections, we have come across two significant facts. One where a PUT primitive may allocate a new block to a file and second, a PUTEOF may delete some blocks from a file. Exactly how is this allocation and deallocation of blocks done ?

In our filing system all the free blocks are linked together in a single chain. The head of this chain is stored in the the first word of the HEADER block. Since a copy of the HEADER block is always kept in memory, a number of disk I/O operations are reduced by reading the address of the next free block directly from the memory.

4.2.11 Descriptor Management

All descriptors are stored in the last 21 tracks (track numbers 179 to 199) on the disk. Their allocation and deallocation is facilitated by a BIT MAP which is kept on block numbers 1 and 2 on track 0. Each descriptor occupies only half a sector. Thus each word of 16 bits in the BIT MAP gives the current status of 16 descriptors in contiguous 8 blocks. If a bit is set, then the descriptor pointed to logically by the position of the bit in the map,

is allocated already to a file. Thus a free descriptor is indicated by a 0 value in the logical bit position for the descriptor.

4.2.12 Addressing Scheme

We have always talked in terms of logical block addresses of the disk sectors. Hence a mapping must be made from the logical address to the actual track, sector and surface address. In this section, the scheme used to form the logical addresses is described.

Each address is one word long. The first seven bits are used to store the logical block number in a cylinder. This number can vary from 0 to 99 since a cylinder has a hundred sectors. The next eight bits store the cylinder number. Since there are 203 tracks, cylinder numbers can vary from 0 to 202. The fifteenth bit (i.e. the most significant bit) is used to locate a descriptor within a block. Since two descriptors are packed into a sector, this bit when set indicates that the specified descriptor is in the latter half of the sector.

Each cylinder has 10 surfaces and 10 sectors in each surface. The block address in the cylinder modulo 10 gives the sector number and the address divided by 10 gives the surface number.

4.2.13 HEADER & ROOT

Both of these are of a half sectorsize and together occupy block zero on track zero. Thus they are both packed into the some sector [DEC]. The Header contains the details of the various disk blocks on the file. Its contents are typically:-

(1) Head of the free blocks chain
(2) Address of the first block storing the BIT MAP for descriptors
(3) Address of the second block
(4) Starting cylinder number for storing descriptors
(5) Last cylinder number used for storing descriptors
(6) Numbers of blocks used for storing descriptors
(7) Starting block address for the Accounting file
(8) Last block address for the Accounting file
(9) Number of blocks used by the Accounting file.

The root is the descriptor of the master file directory of the system. It is created when the system is set up. The owner of this file is the system which has a privileged PPN.

4.2.14 The PPN Scheme

A PPN is one word long. It is divided into 2 fields which together uniquely identify the user. Each field occupies one byte. The first field (i.e. higher byte of the

word) indicates the user's project number and the second field (i.e. the lower byte) indicates the user's programmer number.

Octal numbers are to be used in both cases. Hence each field can range between #0 and #377 [DEC].

4.2.15 System Initialization

To set up the system appropriately an initializing procedure must be followed for the filing system. First the free blocks of the disk must be linked, then the Header, the Root and all the tables must be initialized. The Disk subsystem too must be initialized here.

If and when a crash occurs or the system is shut-down and restarted, the above initializing procedure need not be followed. Here, only the tables and the disk sub-system needs to be re-initialized.

4.2.16 Synchronization

As explained earlier each primitive runs as a part of the user's job. Now when any disk I/O is initiated by the primitive, the user's job is blocked. Another user may now invoke a filing primitive.

Thus all primitives can be invoked concurrently and each may be in a certain stage of execution when blocked. This imposes some restrictions on the filing system to

maintain its integrity. An example of this is given below.

When a free block is allocated or deallocated the contents of the Header block are modified. A user's job may be blocked when the modification is not complete since getting or putting a free block results in more than one disk I/O operations. Another user thus, must not be allowed to access the Header while this modification is going on. Hence with any getting or putting of a block an event number is associated. Binary semaphores are used to lock out this resource and all processes desiring this resource are entered in an event queue [BRIN 73] for the resource.

The same principle is applied in the case of the free descriptor management to preserve the integrity of the BIT MAP. Details of this scheme may be seen in [PAND 82].

There are certain buffers maintained and some global words which reflect the state of the currently invoked request in the case of some filing primitives.

These buffers are mainly used to search a directory for a specific file entry, while the trace of the file is going on in the directory structure.

Here again an event number is associated with this operations and all primitives invoking this operation have to wait on this event. These primitives are OPEN, DELETE, CREATOR, CREDIR, CDPCHK and RENAME.

Thus only one such primitive can be in operation at a time. The other primitives GET, PUT, SEEK, BLKCHK, PUTEOF, CLOSE can run concurrently with each other and with the above six.

4.2.17 Entering of Requests

A filing primitive may often in its execution require to do some disk data transfer operation.

For this purpose, the primitive makes a request via a trap call to the disk-manager. This request is entered into a queue maintained by the disk-manager.

Each request can specify any one of the four functions, (1) Reading of a block, (2) Writing of a block, (3) Reading of a descriptor and (4) Writing of a descriptor. With each request, the block address, the memory location from or to which the data transfer takes place and the job number must be specified. After entering the request the user's job is blocked. It is awakened when the data transfer is over.

CHAPTER 5

DISK MANAGEMENT

This chapter gives a general description of the disk-manager. It also contains a brief account of the disk interrupt servicing routines.

5.1 INTRODUCTION

The disk manager, as has been mentioned earlier, is a part of the kernel. It is designed to control and complete any data transfer operations from or to the disk. Thus all disk accesses must go through the disk-manager.

Mostly disk requests are expected to come from filing system routines. But the system too can ask for any disk service from the disk-manger. The manager maintains a queue of all disk requests and initiates the next one when the current request is over. It also does the useful task of waking up the system/user's job which was blocked for the current I/O.

In the last chapter, there was an account of how requests are entered in the queue by the filing routines. For any other system utility or program, a similar call would be sufficient to invoke the disk-manager. Hence the

entering of requests in the queue is not dealt with in this chapter. We start here from how the I/O operations are actually carried out.

5.2 QUEUE OF REQUESTS

Each request entered in the queue must specify the following:-

1) The job number of the invoking user.

2) The operation desired: This is taken up in more detail in the next section.

3) Memory location: This is the address in memory from or to which the operation is desired. This in most cases must be an 18 bit address, 2 bits of which are stored with the earlier field, as this field is 1 word long only.

4) Disk block specification: Here a block address of the disk sector from or to which I/O must take place is given. The format for this address has been described earlier in Section 4.2.12.

The table maintains two pointers, an IN and OUT pointer. IN points to the next free space available in the table for entering a new request. Thus IN serves as an index to the table when a request is entered. OUT points to the request whose disk I/O operation is going on currently.

IN and OUT are both initialized at zero. A 'PENDING' flag indicates if a disk operation is currently being carried out. If IN is equal to OUT and the PENDING flag is off then a queue empty condition is indicated, otherwise if the PENDING flag is on and IN equals OUT, then a quque full condition is indicated. In the latter case, a user trying to enter a request in the queue is returned an error message.

If the queue is empty then after the first request is entered, the I/O operation is initiated. An interrupt occurs when the operation is complete. The interrupt servicing routine then checks to see if request has been entered meanwhile in the queue, in which case it initiates the next operation. The queue in this sense is largely interrupt driven. On the completion of each request the appropriate job is signalled.

5.3 OPERATIONS

When a request is entered in the queue, any one of the four operations can initially be specified. They are:-

1) Operation 0 - read a block
2) Operation 1 - write a block
3) Operation 2 - read a descriptor
4) Operation 3 - write a descriptor

The first two operations are simply the reading of and writing into a disk sector. Here an 18 bit addrees

must be specified for the date transfer operation to take place with the MAP option in memory. The last two operations are necessary as a descriptor is only of half a sector size. Thus reading and writing of a descriptor from the user's point of view is the transferring of only half a sector.

Hence in both cases, some housekeeping must take place either before or after the operation. Also each whole operation must be indivisible. This implies that no other disk I/O request should be initiated while either of the two operations is going on. Here by the term 'operation' we mean the disk access along with the housekeeping needed. A scheme has been devised to ensure that this integrity is maintained. We shall take up each case separately.

5.3.1 Reading of a Descriptor

This is operation '2' as specified by the user. After the first disk access (i.e. reading of the sector specified by the descriptor's address into a buffer local to disk manager) is initiated, the operation is changed to '4'. The OUT pointer in the table is not incremented. Here the user need only specify a 16 bit relocatable address in his area since the 18 bit address of the local buffer is entered into the controller registers.

When the interrupt servicing starts, in general, the table being interrupt driven, the next request is taken up

and initiated. The next request is always pointed to by OUT. Here the next request is the same one with the operation changed to 4'. So now operation '4' is carried out. That is, the descriptor is transferred to the user's area. After this, the job which had made the request is informed of its completion. The OUT pointer is incremented and the next request is taken up.

5.3.2 Writing of a Descriptor

This is operation '3' as specified by the user. After the first disk I/O operation (i.e. reading of the section into a local buffer) is initiated the operation is changed to '5'. Again the OUT pointer is not incremented.

So the next request to be taken up when this read operation is over, is the same one with operation '5'. Operation '5' transfers the specified descriptor from the users area to the appropriate half of the sector read into the local buffer. The buffer is then written back into the sector. After this, the job is informed of the completion of this request and the next one is taken up.

This, briefly describes the two operations of reading and writing a descriptor. In the case of 'write' too, the user must specify only a 16 bit relocatable address in his area.

5.4 INITIATION OF I/O AND INTERRUPT SERVICING

To initiate the disk I/O operation the appropriate disk controller registers must be filled and the command to be given, entered in the controller.

For this a current request table is maintained which keeps track of the operation on at present. In this table only a 'read sector' or a 'write sector' operation can be specified. The purpose of this table is mainly to allow retries (upto a maximum of five tries can take place) of the same operation in case an error occurs.

After the operation is completed, the interrupt servicing starts. The interrupt servicing routine drives the queue. At each interrupt it checks for the correct completion of the operation. In cases of errors, it initiates the next try and keeps a count of the number of retries. Otherwise it wakes up the job blocked on this disk I/O request and takes up the next request in the queue. In the case of operations 2 and 3, the waking up of the job is postponed to until 4 and 5 respectively are over. If there is no further request in the queue, it resets the PENDING flag and returns.

Here non-data transfers are not initiated separately, hence the only case in which a non-data interrupt servicing is needed, is if an error occurs in the seeking or searching

of a track and sector. This interrupt routine then checks for the error, retries the operation if possible and returns. As the operation is always a data transfer operation the next interrupt (if the operation completes successfully) should be serviced by the data transfer servicing routine.

5.5 CONCLUSION

The model for the disk-manager has been described in this chapter. A necessary optimization could be made in the handling of the driver (i.e. initiation of operation and interrupt servicing) by checking and handling each kind of error appropriately. At present, all errors are treated in a similar fashion and after 5 retries, the request is aborted.

CHAPTER 6

THE LINE PRINTER SPOOLER

This chapter deals with the spooler to print files. It first gives a general description of the spooler process and later of the line printer driver.

6.1 THE SPOOLER PROCESS

The spooler runs as a user process. Other user's communicate with it to get their files printed. This communication is through a call 'ENTERSPOOL' which enters the user's request into a queue maintained by the spooler [SHAW 79].

The spooler first activates the next file in the output spool list by invoking the file primitive OPEN. The file header record "the Banner", which is essentially the user's PPN, is printed to help users to identify their files. Then blocks of the file are read successively (by invoking the filing primitive GET), into a buffer kept in the spooler area. After a file block is fetched, it is transferred to a ring of buffers of one line size each maintained by the line printer driver. Only after all the characters in the block are transferred is the next GET

invoked.

The line printer driver then initiates the printing of each input full buffer. The spooler process may be blocked on two events. (1) When it waits for a disk operation to be completed. This occurs when it invokes any one of the file primitives OPEN and GET or even CLOSE. After all the blocks of the file have been transferred CLOSE must be invoked to deactivate the file. (2) When it waits for an empty one line buffer. This may occur if the buffers are filled up faster than they are emptied (i.e. printed).

6.1.1 The Spool Table

The spooler maintains a queue of all requests for the printing of files. With each request the user must specify (1) his job number (2) pathname of the file (3) the user's PPN (4) page limit for the file. A sequence number is added by the spooler. This number uniquely identifies a request in the queue and may be used later to kill or modify a request or see its status. Since only three words are available for storing pathnames, the user must specify the path from the directory which contains the relevant file. There is provision for only one file entry per request. SIN is used to enter the next request and SOUT to remove the next request from the queue for printing [MADN 74].

6.2 LINE PRINTER DRIVER

The line printer maintains a ring of15 one line buffers. Each line can have a maximum of 132 characters. The buffer size has been kept at 134 characters to store end of line and end of file characters too in the buffer. Buffers are of 3 kinds:- IN, OUT and EM (i.e. input full buffers, output buffers and empty buffers respectively). There is only one output buffer at any time which is currently being printed. [SHAW 74] .

A single chain is used to link all input buffers together and another single chain to link all empty buffers. Pointers to the head and tail of each chain are maintained. Each buffer is referenced by a number.

Two routines ADDBUF and TAKEBUF are called to add or remove a buffer from a chain. Semaphores are used to keep a count of the number of buffers in each chain and synchronize the allocation and deallocation of buffers.

The primitive SENDLINE is invoked by the spooler to get an empty buffer and fill in a line and add it to the input buffers chain. PUTLINE is called by the driver to initiate the printing of the line contained in the first buffer in the input chain. A count of the number of lines printed per page is maintained to help control the movement of paper.

This in brief gives a general description of the spooler. The programs have been listed in the appendix. Due to a shortage of time, the printing of the banner has been restricted to the PPN only. An improvement can be made by including the filename and date too. Also the printing of banners in giant letters could be incorporated.

CHAPTER 7

CONCLUSION

In this chapter a brief description of the accounting procedure is given. There is a section on the error handing. The last section suggests improvements that could have been made and gives an idea of the further work possible in the system.

7.1 ACCOUNTING

This has not been implemented. As such this section serves as a guide to the accounting procedure possible.

For each user an account record is maintained which contains:- (1) His PPN (2) PASSWORD (3) Disk Log-Out Quota, (4) Connect Time and (5) Disk Quota Used. All the accounting records are stored on specified disk blocks. Space has been allocated for them from blocks 4 to 99 on cylinder 0. This information is stored in the Header.

The accounting blocks do not constitute a file and thus do not have a descriptor. They are accessed by invoking any one of the four primitives:-

(1) INSERT (PPN)

(2) REMOVE (PPN)

(3) MODIFY (PPN, FIELD, NEWVALUE)

(4) EXAMINE (PPN)

INSERT adds a new PPN and the corresponding record into the specified block. REMOVE deletes a PPN with its record. MODIFY will change the value of the FIELD specified to the NEWVALUE. Examine returns the value of the FIELD.

Given the PPN a hashing function can be used to identify the required disk blocks amongst those allocated for the accounting records. A hashing function has been suggested below.

The PPN has 2 fields, a project number and a programmer no. each varying between #0 and #377. The project field modulo 64 can be used to refer to the first 64 blocks. The PPN can now be searched for in the block obtained. Overflows from the blocks are stored in the last 32 blocks.

The primitive MODIFY will have to be incorporated into two filing routines CLOSE and DELETE to maintain the integrity of the number of disk blocks owned by the PPN.

7.2 ERROR HANDLING

Error handling is of two kinds. In one case an error value is returned to the routine invoking the filing primitive and in the other case the user's job is aborted,

his stack reinitialized and an error message displayed.

The error handling is not complete. Appropriate messages have to be displayed. Appendix III contains a list of error messages. All error values begin from the number #077000. The values associated with the messages are added to this number.

7.3 REMARKS

The filing system and the spooler have been described in the earlier chapters. Regarding both of them, some remarks have been included here. These include both suggested improvements in the system and an account of its current status.

(1) Although all the filing routines have been implemented, to make them usable, a number of system commands must be written. Examples of these are LOGIN, PRINT, TYPE, COPY etc.

(2) Another incomplete portion is the printing of suitable BANNERS in giant letters by the spooler. Currently only the user's PPN is printed.

(3) One of the factors which increased the complexity of the system was the execution of the filing primitives in user mode and the disk-manager in

the supervisor mode. Inter-mode communication of buffer addresses had to be incorporated.

(4) Another significant point is the half a sector size descriptor. Possibly descriptors occupying a full block each would have been less cumbersome.

(5) Buffer management is not a part of the system. A user can also specify only one buffer for each open file. This is restrictive since 2 blocks of the file cannot be in memory at the same time. Also routines to get and put characters and other status information for the buffer must be kept by each user. A common buffer pool could have localized this management.

The design, otherwise is extremely simple and general in nature to ensure that modifications can easily be made. Program listings are contained in the Appendix IV.

REFERENCES

1. [BRIN 73] Brinch Hansen, P., "OPERATING SYSTEM PRINCIPLES", Chapter 3, 1973, Prentice Hall, Inc.

2. [MADN 74] Madnick and Donovan, "OPERATING SYSTEMS", Chapter 5 and 6, 1974, McGraw-Hill Kogakusha Ltd.

3. [PANO 82] Pandya, P.K., "A TIME-SHARING SYSTEM FOR TDC-316", M.Tech. Thesis, Computer Science Programme, Indian Institute of Technology, Kanpur, 1982.

4. [RITC 78a] Ritchie and Thompson, "UNIX TIME-SHARING SYSTEM", The Bell Systems Technical Journal, Vol. 57, No. 6, Part 2, 1978, pp 1905-1929.

5. [RITC 78b] Ritchie, D.M., "UNIX TIME-SHARING SYSTEM: A RETROSPECTIVE", The Bell Systems Technical Journal, Vol. 57, No. 6, Part 2, 1978, pp 1947-1967.

6. [SHAW 74] Shaw, A.C., "THE LOGICAL DESIGN OF OPERATING SYSTEMS", Chapters 3 and 9, Prentice-Hall, Inc.

7. [SHAW 79] Shaw, A.C., Design of a Single Language Multiprogrammer System in "PRINCIPLES OF SOFTWARE ENGINEERING & DESIGN" by Zelkowitz, Shaw and Gannon, Chapter 4, 1979, Prentice-Hall, Inc.

8. [THOM 72] Thomas J.R., The Structure of a Time-Sharing System in "OPERATING SYSTEM TECHNIQUES", pp 351-370, 1972, Ed. Hoare and Perrott, Academic Press.

9. [WILS 76] Wilson, R., The Titan Supervisor, in "STUDIES IN OPERATING SYSTEMS", Chapter 4, 1976, McKeag and Wilson, Academic Press.

10. [TDC] "SYSTEM MANUAL TDC-316", Vol. I, II and III, Electronics Corporation of India Limited, Hyderabad.

11. [DEC] "DEC-10 MANUAL, MONITOR STRUCTURE", pp 44-55, Digital Equipment Corporation.

APPENDIX I

VALUE RETURNED TO THE USER		LIST OF PRIMITIVES
1. CHANNEL NUMBER	=	CREATOR(JOBNO,PPN,PATHNAME, PROTECTION,BUFFER)
2. CHANNEL NUMBER	=	OPEN(JOBNO,PPN,PATHNAME, FUNCTION,BUFFER)
3. MESSAGE OF SUCCESSFUL COMPLETION	=	CREDIR(JOBNO,PPN,PATHNAME, PROTECTION,OWNER)
4. DIRECTORY ADDRESS	=	CDPCHK(JOBNO,PPN,PATHNAME)
5. CURRENT VALUE OF THE BLOCK POINTER	=	GET(JOBNO,CHANNEL NUMBER)
6. CURRENT VALUE OF THE BLOCK POINTER	=	PUT(JOBNO,CHANNEL NUMBER)
7. CURRENT VALUE OF THE BLOCK POINTER	=	SEEK(JOBNO,CHANNEL NUMBER, BLOCK NUMBER)
8. CURRENT VALUE OF THE BLOCK POINTER	=	BLKCHK(JOBNO,CHANNEL NUMBER)
9. NUMBER OF BLOCKS FREED	=	PUTEOF(JOBNO,CHANNEL NUMBER)
10. MESSAGE OF SUCCESSFUL COMPLETION	=	CLOSE(JOBNO,CHANNEL NUMBER)
11. NUMBER OF BLOCKS FREED	=	DELETE(JOBNO,PPN,PATHNAME)
12. MESSAGE OF SUCCESSFUL COMPLETION	=	RENAME(JOBNO,PPN,PATHNAME, NEW PROTECTION)

NOTE: IN THE CASE OF RENAME , THE NEW FILE NAME IS SPECIFIED IN THE PATHNAME AFTER THE END OF LIST MARKER .

APPENDIX II

ACTIVE TABLE

Each table entry consists of the followins fields:-

	FIELDNAME	POSITION
1.	Job Number	First Word
2.	Current Block Pointer	Second Word
3.	Status Word	Third Word
4.	File opened for READ or READ-WRITE	Bit 0 in Status Word
5.	Current descriptor modified	Bit 2 in Status Word
6.	First descriptor modified	Bit 1 in Status Word
7.	Current descriptor number	Bits 3 to 7 in Status Word
8.	File is in DELETE state	Bits 8 to 13 in Status Word
9.	Memory extension bits of the address of the buffer used by the file.	Bits 14 & 15 in Status Word
10.	Size of file	Fourth Word
11.	The first descriptor's address for the file	Fifth Word
12.	Address of buffer to be for the file.	Sixth Word

APPENDIX III

ERROR MESSAGES

ERROR NO.	MESSAGE
0	ERROR ON DISK ACCESS DATA TRANSFER
1	ILLEGAL FILE SPECIFICATION
2	FILE DOES NOT EXIST
3	FILE ALREADY EXISTS
4	FILE IS NOT A DIRECTORY FILE
5	ILLEGAL WRITE ACCESS ON DIRECTORY OPENING
6	FILE OPENED EARLIER
7	FILE OPENED FOR WRITE EARLIER
8	PROTECTION FAILURE
9	ACTIVE FILES CAPACITY EXCEEDED
10	ILLEGAL CHANNEL SPECIFICATION
11	WRITE ACCESS INVALID ON A FILE
12	ERROR!! ATTEMPT TO READ BEYOND EOF
13	DRIVE FAULT
14	NO FREE BLOCK AVAILABLE
15	NO FREE DESCRIPTOR AVAILABLE
16	DIRECTORY FILE TO BE DELETED ONLY WHEN EMPTY
17	DISK REQUEST CAPACITY EXCEEDED

% LIST OF ORDINARY MESSAGES %

NO.	MESSAGE
-1	END OF FILE REACHED

```
00010   MODULE  FILINGSYSTEM(NOLIST) =
00020   begin
00030
00040           % in this file all global declarations of all
00050             variables and tables used by the filing
00060             system have been made     %
00070
00080           %    author      :  AARTI  KUMAR
00090            creation date : 5 MAY ,1982      %
00100
00110
00120       MACRO
00130                   JOBNO           = 0$,    ! user's job number
00140                   FILENAME        = 0$,    ! filename field
00150                   CURBLK          = 1$,    ! current block pointer
00160                   FILSTAT         = 2$,    ! file status word
00170                   SIZE            = 3$,    ! size of file
00180                   PARENT          = 4$,    ! address of directory
00190                   PREVDESC        = 4$,    ! previous descriptor's address
00200                   CURDESC         = 5$,    ! current descriptor's address
00210                   NEXTDESC        = 6$,    ! next descriptor's address
00220                   BUFADDR         = 5$,    ! address of buffer in user's area
00230                   OWNER           = 7$,    ! owner's ppn
00240                   DATE            = 8$,    ! date of creation of file
00250                   STORAGE         = 10$,   ! storage map for file starts
00260                   descno          = 3,5$,  ! descriptor number
00270                   delbits         = 8,6$,  ! indicates if > 1 - file to be deleted
00280                   extension       = 14,2$, ! stores 2 bits of 18 bit address
00290                   curmod          = 2,1$,  ! current descriptor modified
00300                   firstmod        = 1,1$,  ! first descriptor modified
00310                   wrtbit          = 0,1$,  ! (0/1- file opened for read/read-write)
00320                   kind            = 9,1$,  ! ( 0/1- text/directory file)
00330                   protection      = 0,9$,  ! file protection
00340                   latter          = 15,1$, ! specifies position of descriptor
00350                   oprn            = 0,4$,  ! function for diskio
00360                   MAXM            = 10$,   ! maximum size of active table
00370                   HEADER          = #000000$, ! address of header
00380                   ROOT            = #100000$, ! address of root
00390                   ERRVAL          = #077000$, ! error range starting value
00400                   MAXERR          = #077700$, ! maximum error value
00410                   EOLIST          = '$'$;  ! end of list marker
00420
00430   STRUCTURE   TABLE[CH,FIELD] =
00440           [CH*FIELD*BYTES](.TABLE + .CH * FIELD * BYTES + .FIELD * BYTES)<0,8*BYTES>;
00450
00460   word global TABLE ACTIVE[6*10]   ,          ! ACTIVE TABLE FOR ALL OPEN FILES
00470                TABLE  DESCRIPTOR[64*10];  ! DESCRIPTORS OF ALL OPEN FILES
00480
```

```
00530     global  PATH      ,     ! stores pointer to pathname
00540             ADDRESS   ,     ! stores directory's address
00550             CHANNEL   ,     ! stores current channel number allocated
00560             STORE     ,     ! stores address of block containing file entry
00570             INDEX     ,     ! stores pointer to storage map of file
00580             FDERROR   ,     ! stores error value when routine exits
00590             NJOB      ;     ! stores current job's number
00600
00610     external ENTEREQUEST;  ! this is called to enter request in disk queue
00620
00630     MACRO OUTERR(NUM)=
00640             ( FDERROR =    (NUM + ERRVAL) ; RETURN -1)$;
00650
00660     GLOBAL ROUTINE FILINIT=
00670       BEGIN
00680             % initialises all filing global variables %
00690
00700          ADDRESS = PATH = INDEX = 0;
00710      CHANNEL = STORE = FDERROR = NJOB =0;
00720             INCR I FROM 0 TO 127 DO ( LINEAR[.I] = 0 );
00730             INCR I FROM 0 TO 63 DO ( TRACEBUF[.I] = 0 );
00740        END;
00750
00760     GLOBAL ROUTINE DISKIO(FUNC,DSKADDR,MEMLOCN)=
00770      begin
00780        % this routine enters request for diskio %
00790
00800       Local TASK,NEWLOCN;
00810
00820             TASK<oprn> = .FUNC;
00830             if .FUNC leq 1 then
00840                     begin
00850                             NEWLOCN = .MEMLOCN;
00860                     end;
00870                     NEWLOCN = .MEMLOCN;
00880             ENTEREQUEST(.NJOB,.TASK,.DSKADDR,.NEWLOCN);
00890      end;
00900
00910
00920             %  MODE OF ACCESS OF DECRIPTOR FIELDS: [CHANNEL , <fieldname> ]
00930             MODE OF ACCESS OF FIELDS IN ACTIVE TABLE: [ CHANNEL, <fieldname> ] %
00940
00950     end  ELUDOM
```

```
00010
00020          % this file is required by all filing primitives %
00030
00040          %   author      :  AARTI  KUMAR
00050           date of creation :  5 MAY,1982  %
00060
00070      MACRO
00080                  JOBNO           = 0$,       ! user's job number
00090                  FILENAME        = 0$,       ! filename specified
00100                  CURBLK          = 1$,       ! current block pointer
00110                  FILSTAT         = 2$,       ! file status word
00120                  SIZE            = 3$,       ! size of file
00130                  PARENT          = 4$,       ! directory's address
00140                  PREVDESC        = 4$,       ! previous descriptor's address
00150                  CURDESC         = 5$,       ! current descriptor's address
00160                  NEXTDESC        = 6$,       ! next descriptor's address
00170                  OWNER           = 7$,       ! file owner's ppn
00180                  DATE            = 8$,       ! creation date of file
00190                  STORAGE         = 10$,      ! storage map starts
00200                  BUFADDR         = 5$,       ! user's buffer in memory (address)
00210                  oprn            = 0,4$,     ! function for diskio
00220                  descno          = 3,5$,     ! descriptor number
00230                  delbits         = 8,6$,     ! ( file to be deleted if >0)
00240                  extension       = 14,2$,    !  stores 2 extra bits of address
00250                  curmod          = 2,1$,     ! current descriptor modified
00260                  firstmod        = 1,1$,     ! first decriptor modified
00270                  wrtbit          =0,1$,      ! file opened for write if bit 1
00280                  kind            = 9,1$,     ! ( 0/1 = text/directory file)
00290                  protection      = 0,9$,     ! protection for file
00300                  latter          = 15,1$,    ! specifies position of descriptor
00310                  MAXM            = 10$,      ! specifies number of open files
00320                  HEADER          = #000000$, ! header address
00330                  ROOT            = #100000$,  ! root directory's address
00340                  ERRVAL          = #077000$,  ! error value ( starting)
00350                  MAXERR          = #077700$,  ! maximum error value possible
00360                      NUM         = 6$,        ! no of fields in active
00370                      IND         = 64$,       ! no of words in descriptor
00380                  EOLIST          = 0$;        ! end of list marker
00390
00400      structure   TABLE[CH,FIELD] =
00410            (.TABLE + .CH * NUM * 2  + .FIELD * 2)<0,8*2>;
00420      structure   ARRY[CHNL,PTR] =
00430            (.ARRY + .CHNL * IND * 2 + .PTR *2 )<0,8*2>;
00440
00450      external ACTIVE , DESCRIPTOR  ,   HEADBLK  ;
00460      map TABLE ACTIVE ;    ! active table for all open files
00470      map ARRY DESCRIPTOR;   ! descriptors of all open files stored
00480      map VECTOR HEADBLK ;   ! stores header block in memory
```

```
00530       ENTEREQUEST - enters the diskio request in disk manager's queue
00540       PROTCHK - does a protection check on file   }    %
00550
00560     external GETBLK,PUTBLK;  ! gets and puts free blocks on disk
00570     external GETDESC,PUTDESC;  ! allocates and releases free descriptors
00580     external WAITSEM,SIGNALSEM;
00590         ! synchronizes the blocking and awakening on events
00600
00610
00620     MACRO OUTMSG(NUMB)=
00630             (RETURN (NUMB))$;
00640
00650
00660
00670     % mode of access of tables ACTIVE & DESCRIPTOR :=
00680                   [ <channel number> , < fieldname>    ] ;    %
00690
```

```
00010     MODULE  SETUPINIT(NOLIST)=
00020      begin
00030             % this file contains two initializing routines
00040              (1) SYSINIT: this is called when the system
00050                    is set up
00060              (2) TMPINIT: this is called every time the
00070                    system is restarted    %
00080
00090             %       author       :  AARTI  KUMAR
00100              date of creation    :  24 JULY, 1982     %
00110
00120       MACRO
00130             HEADER  = #000000$;    ! block address of header
00140
00150       external INITROOT    ,  ! initializes the root directory
00160                INITSTORAGE ,  ! creates a linked list of all free disk blks
00170                HEADINIT    ,  ! initializes the header block
00180                MAPINIT     ,  ! initializes the free descriptor map
00190                FILINIT     ,  !  initializes all global variables and arrays
00200                INITACTIVE  ,  !  initializes tthe active table
00210                INITSTAT    ,  !  initializes the table maintained by disk manager
00220                INITDISK    ,  !  initializes the disk controller
00230                DISKIO      ;  !  called to enter a disk-io request
00240
00250       external  NJOB;
00260       external HEADBLK;
00270       map VECTOR  HEADBLK;
00280
00290       GLOBAL ROUTINE SYSINIT(JOB)=
00300         begin
00310             % this routine is called when the system is set up %
00320
00330             NJOB = .JOB;
00340             INITDISK();
00350             INTSTAT();
00360             FILINIT();
00370             INITSTORAGE();
00380             INITROOT();
00390             MAPINIT();
00400             HEADINIT();
00410       end;
00420
00430     GLOBAL ROUTINE TMPINIT(JOB)=
00440       begin
00450             % this routine is called each time the system restarts %
00460
00470             NJOB = .JOB;
00480             INITDISK();
00490             INTSTAT();
00500             INITACTIVE();
00510             FILINIT();
00520             DISKIO( 2,HEADER, HEADBLK);
```

```
00530       end;
00540
00550
00560     end  ELUDOM
```

```
00010     MODULE INITIALIZE(NOLIST)=
00020     begin
00030           % routine to initialize disk space %
00040
00050           %       author       :  AARTI  KUMAR
00060             date of creation : 10 MAY , 1982     %
00070
00080         MACRO
00090           FILENAME = 0$,              !  field : <filename>
00100              JOBNO = 0$,              !  field : <jobno>
00110            FILSTAT = 2$,              !  <file status word>
00120               SIZE = 3$,              !  <size of file >
00130             PARENT = 4$,              !  descriptor address of directory
00140           PREVDESC = 4$,              !  decriptor address of previous
00150            CURDESC = 5$,              ! address of current descriptor
00160           NEXTDESC = 6$,              ! address of next descriptor
00170              OWNER = 7$,              ! file owner's ppn
00180               DATE = 8$,              ! file date of creation
00190            STORAGE = 10$,             ! storage map of file starts
00200               MAXM = 10$,             ! maximum number of acyive files
00210               kind = 9,1$,            ! { 0/1 } - directory/text file
00220           protection = 0,9$,          ! protection for file
00230           TRKFULL  = 82$,             ! number of tracks in disk
00240           CYLFULL  = 9$,              ! number of blocks in a cylinder
00250           HEADER  =#000000$,          ! address of header block
00260           ROOT = #100000$,            ! address of root directory
00270           latter   = 15,1$,           ! bit indicates position of descriptor in block
00280           BLANK    =  ' '$,           ! {blank character}
00290           trk = 7,8$,                 ! track address field
00300           cylinder = 0,7$;            ! cylinder address {logical} field
00310
00320
00330     STRUCTURE TABLE[ch,field] =
00340           (.TABLE + .ch * 6 * 2  + .field * 2)<0,8*2>;
00350     STRUCTURE ARRY[DPNT,DFIELD] =
00360           (.ARRY + .DPNT * 64 * 2  + .DFIELD *  2)<0,8*2>;
00370
00380     external ACTIVE,DESCRIPTOR;
00390     external GDATE;
00400     map TABLE ACTIVE;
00410     MAP ARRY DESCRIPTOR;
00420
00430     external HEADBLK;
00440     map VECTOR HEADBLK;
00450
00460       external DISKIO, INTSTAT, INITDISK;
00470       external NJOB;
00480
00490     external TRACEBUF,LINEAR;
00500     map VECTOR TRACEBUF:LINEAR;
00510
00520     GLOBAL ROUTINE INITROOT=
```

```
00530     begin
00540        Local VALUE;
00550              %  this routine initializes the root     %
00560
00570              DISKIO(2,ROOT,TRACEBUF);
00580              TRACEBUF[ FILENAME] = 'RO';
00590              TRACEBUF[ FILENAME + 1 ] = 'OT';
00600              TRACEBUF[ FILSTAT] <kind> = 1;
00610              TRACEBUF[FILSTAT]<protection> = #022;
00620              TRACEBUF[ SIZE] =0;
00630              TRACEBUF[ PARENT] = ROOT;
00640              TRACEBUF[ CURDESC] = ROOT;
00650              TRACEBUF[ NEXTDESC] = -1;
00660              TRACEBUF[ OWNER ] = #000401 ;
00670              TRACEBUF[ DATE  ] = GDATE();
00680              TRACEBUF[STORAGE] = '$';
00690              DISKIO(3,ROOT,TRACEBUF);
00700     end;
00710
00720
00730     GLOBAL ROUTINE INITSTORAGE =
00740     begin
00750     Local  NEXT,CURRENT,BLK,I,J;
00760              % this creates a linked list of all free blocks on disk  %
00770
00780              I =80;
00790              J= 0; BLK =0;
00800              BLK<trk> = .I;
00810              BLK<cylinder> =.J;
00820              CURRENT = .BLK;
00830              NEXT = .BLK + 1;
00840              INCR L FROM 0 TO 127 DO  (LINEAR[.L] =0 );
00850              do
00860              begin
00870                       LINEAR[0] = .NEXT;
00880                       DISKIO(1,.CURRENT,LINEAR);
00890                       CURRENT = .NEXT ;
00900                        NEXT = .NEXT + 1;
00910                       if .NEXT<cylinder> eql CYLFULL then (NEXT<trk> = .NEXT<trk> +1;
00920                                                 NEXT<cylinder> = 0);
00930              end
00940              until .NEXT<trk> eql (TRKFULL );
00950              NEXT = HEADER;
00960              LINEAR[0] = .NEXT;
00970              DISKIO(1,.CURRENT,LINEAR);
00980              LINEAR[0] = .BLK;
00990              DISKIO(1,HEADER,LINEAR);
01000     end;
01010
01020     GLOBAL ROUTINE INITACTIVE=
01030     begin
01040        Local J;
```

```
01050
01060           % this initializes the active table  %
01070
01080           J = 0;
01090           do  begin
01100                   INCR I FROM 1 TO 5 DO (ACTIVE[.J,.I] =0);
01110                   ACTIVE[.J,0] = -1 ; J = .J+1;
01120                           INCR I FROM 0 TO 63 DO
01130                                   (DESCRIPTOR[.J,.I] = 0);
01140           end until (.J eql MAXM);
01150   end;
01160
01170   GLOBAL ROUTINE HEADINIT=
01180    begin
01190
01200           % this initializes the header block %
01210
01220       DISKIO(2,HEADER,HEADBLK);
01230        HEADBLK[1] = ROOT;              ! address of root directory
01240        HEADBLK[2] = #000001;           ! address of 1'st block of descriptor map
01250        HEADBLK[3] = #000002;           ! address of 2'nd block of descriptor map
01260        HEADBLK[4] = 179;               ! starting track number of decriptors
01270        HEADBLK[5] = 199;               ! ending track number of descriptors
01280        HEADBLK[6] = 4096;              ! number of blocks used to store descriptors
01290        HEADBLK[7] = #000004;           ! starting block number of accounting file
01300        HEADBLK[8] = #000143;           ! ending block number of accounting file
01310        HEADBLK[9] = 96;                ! number of blocks in accounting file
01320        DISKIO(3,HEADER,HEADBLK);
01330    end;
01340
01350   END ELUDOM
```

```
00530          end;
00540
00550
00560      GLOBAL ROUTINE PUTBLK(JOB,VALUE) =
00570         begin
00580              Local  ADDR2;
00590              % put a free block  %
00600
00610              WAITSEM(1); PJOB = .JOB;
00620                ADDR2 =.HEADBLK[0];
00630                HEADBLK[0]  =.VALUE;
00640              BLKIO(3,HEADER,HEADBLK);
00650                BUFF1[0]  = .ADDR2;
00660              INCR I FROM 1 TO 127 DO ( BUFF1[.I] = ZERO );
00670              BLKIO(1,.VALUE,BUFF1);
00680              SIGNALSEM(1);
00690         end;
00700
00710
00720      end ELUDOM
```

```
00010     MODULE BITMAP(STACK,NOLIST)=
00020     begin
00030
00040             % this module contains routines for free descriptor management    %
00050
00060             %       AUTHOR          :  AARTI KUMAR
00070              DATE OF CREATION    : 15 MAY , 82       %
00080
00090        BIND
00100             Header = #000000,          ! address of header block
00110              Addr1 = #000001,          ! address of first block of BIT MAP
00120              Addr2 = #000002;          ! address of second block of BIT MAP
00130
00140        MACRO
00150                ERRVAL = #077000$,
00160                 oprn  = 0,4$,
00170                FILBLK = 0,7$,
00180                TRKCYL = 7,8$,
00190                latter = 15,1$,
00200                lowlim = 179$,
00210               upperlim = 199$;
00220
00230     global BJOB;
00240
00250     MACRO  OUTBIT(DIGIT)[]=
00260           if not .BUFF2[.I]<$LENGTH-1,1> then ( FLAG = 1;   BUFF2[.I]<$LENGTH-1,1> = 1)
00270                       else ( COUNT = .COUNT+1;  OUTBIT($REMAINING))$;
00280
00290     MACRO FILLBIT(VAL)[]=
00300             if .BIT eql ($LENGTH-1) then ( (.TEMP)<$LENGTH-1,1> = 0 )
00310                       else  ( FILLBIT($REMAINING))$;
00320
00330     external ENTEREQUEST ,DIVL,MODL;
00340     external WAITSEM,SIGNALSEM;
00350
00360
00370     GLOBAL ROUTINE DESCIO(FUNC,DSKADDR,MEMLOCN)=
00380       begin
00390
00400        % this routine enters request for diskio %
00410
00420       Local TASK,NEWLOCN;
00430
00440             TASK<oprn> = .FUNC;
00450             if .FUNC leq 1 then
00460                       begin
00470                                 NEWLOCN = .MEMLOCN;
00480                       end;
00490                       NEWLOCN = .MEMLOCN;
00500             ENTEREQUEST(.BJOB,.TASK,.DSKADDR,.NEWLOCN);
00510      end;
00520
```

```
00530     word own VECTOR BUFF2[128];
00540
00550     GLOBAL ROUTINE GETDESC(JOB)=
00560       begin
00570             Local  TRKNO,COUNT,TEMP,ADDR,FLAG,BLKNO,WRD,BIT,I,VAL;
00580
00590              % gets a free descriptor %
00600
00610              WAITSEM(2) ; BJOB = .JOB;
00620              FLAG = 0;   COUNT = 0;  VAL = 0;
00630              ADDR = ADDR1;
00640              DESCIO(0,ADDR1,BUFF2);  I = 0;
00650              while ((.COUNT leq 4095 )  and   (not .FLAG ))
00660              do begin
00670                       TEMP = BUFF2[.I];
00680                       OUTBIT(16,15,14,13,12,11,10,9,8,7,6,5,4,3,2,1);
00690                       I = .I +1;
00700                       if (not .FLAG and .I eql 128 )  then
00710                                (  ADDR = ADDR2; DESCIO(0,ADDR2,BUFF2); I = 0);
00720              end;
00730              if (not .FLAG )  then  (SIGNALSEM(2); RETURN -1);
00740              DESCIO(1,.ADDR,BUFF2);
00750              TRKNO = lowlim + DIVL(.COUNT,200);
00760              BLKNO = DIVL(  MODL(.COUNT,200) ,2 );
00770              VAL = 0;
00780              VAL<FILBLK> = .BLKNO;
00790              VAL<TRKCYL> = .TRKNO;
00800              if MODL(.COUNT , 2) eql  1 then (VAL = .VAL + #100000);
00810              INCR L FROM 0 TO 63 DO ( BUFF2[.L] =0);
00820              DESCIO(3,.VAL,BUFF2);
00830              SIGNALSEM(2);
00840         .VAL
00850     end;
00860
00870     GLOBAL ROUTINE PUTDESC(JOB,VALUE) =
00880     begin
00890             Local  TRKNO,COUNT,TEMP,ADDR,FLAG,BLKNO,WRD,BIT,I;
00900
00910              % this routine puts a free descriptor  %
00920
00930
00940
00950              WAITSEM(2);  BJOB = .JOB;
00960              TRKNO = 0; BLKNO =0;
00970              BLKNO = .VALUE<FILBLK> ;
00980              TRKNO = .VALUE<TRKCYL> ;
00990              COUNT = (.TRKNO - lowlim) * 200 + .BLKNO  * 2;
01000              if .VALUE<latter> then COUNT = .COUNT + 1;
01010              if .COUNT leq 2045 then (ADDR = ADDR1) else (ADDR = ADDR2);
01020              DESCIO(0,.ADDR,BUFF2);
01030              WRD = DIVL( .COUNT , 16);
01040              BIT = 15 -  MODL( .COUNT , 16);
```

```
01050               TEMP = BUFF2[.WRD];
01060               FILLBIT(16,15,14,13,12,11,10,9,8,7,6,5,4,3,2,1);
01070               DESCIO(1,,ADDR,BUFF2);
01080               SIGNALSEM(2);
01090       end;
01100
01110       external DISKIO;
01120       GLOBAL ROUTINE MAPINIT=
01130        begin
01140
01150               Incr I from 0 to 127 do
01160                     begin BUFF2[.I] = 0 end;
01170               DESCIO(1,ADDR1,BUFF2);
01180               DESCIO(1,ADDR2,BUFF2);
01190        end;
01200
01210       end ELUDOM
```

```
00010     MODULE MAKEDIRECTORY(NOLIST)=
00020     begin
00030
00040             %         AUTHOR          :  AARTI KUMAR
00050               DATE OF CREATION    :  JUNE 5, 1982 %
00060
00070             % this primitive is used to create directory files
00080               a message of successful completion is returned
00090               unlike the creation of text files the file is not
00100                 activated here     %
00110
00120
00130     require FILER1.B11;
00140     require FILER2.B11;
00150
00160     external GDATE,INITDESC,ENTERVAL,FINDSPACE;
00170
00180     ROUTINE NEWDIR(ADDR,PATH,CHKLOCK,OWNR)=
00190             % this routine initializes the descriptor for a new
00200               directory file      %
00210
00220         begin
00230             DISKIO(2,.ADDR,TRACEBUF);
00240                     TRACEBUF[  FILENAME] =.(.PATH);
00250                     TRACEBUF[  FILENAME + 1] =.(.PATH +2);
00260                     TRACEBUF[FILSTAT]<kind> = 1;
00270                     TRACEBUF[FILSTAT]<protection> = .CHKLOCK;
00280                     TRACEBUF[  SIZE] = 0;
00290                     TRACEBUF[ PARENT ] = .ADDRESS;
00300                     TRACEBUF[  CURDESC] =.ADDR;
00310                     TRACEBUF[  NEXTDESC] = -1;
00320                     TRACEBUF[  OWNER] = .OWNR;
00330                     TRACEBUF[ DATE  ] = GDATE();
00340                     TRACEBUF[  STORAGE] =EOLIST;
00350             DISKIO(3,.ADDR,TRACEBUF);
00360         end;
00370
00380
00390     GLOBAL ROUTINE CREDIR(JOB, PPN,FILNAM,CHKLOCK,OWNR)=
00400     begin
00410       Local  CURRENT,TEMP,PREVIOUS;
00420
00430             % this is the file primitive "credir" used to creat
00440               directory files  . unlike 'creator' the owner of
00450               the directory must be given as a primitive %
00460
00470       begin
00480             WAITSEM(0);  ! check to wait on event 0
00490             NJOB= .JOB; FDERROR = 0;
00500             if .(.FILNAM) eql -1 then ADDRESS=ROOT else ADDRESS=.(.FILNAM);
00510             PATH=.FILNAM + 2;  TEMP =.NJOB;
00520             ADDRESS =TRACE(.ADDRESS,.PPN);  ! locate parent directory's address
```

```
00530           if (.ADDRESS gtr ERRVAL  )  and (.ADDRESS lss MAXERR )
00540                 then  ( SIGNALSEM(0) ; RETURN .ADDRESS );  ! illegal path
00550           if not PROTCHK(.PPN,2,.TRACEBUF[OWNER],.TRACEBUF[FILSTAT] )
00560                 then  ( SIGNALSEM(0) ; OUTERR(8) ) ;  ! file protected
00570           CURRENT  = SEARCHDIREC(.ADDRESS);  ! search for file in directory
00580           if .CURRENT neq -1 then  ( SIGNALSEM(0) ; OUTERR(3) );
00590                 ! file already exists
00600           DISKIO(2,.ADDRESS,TRACEBUF);
00610           if not PROTCHK(.PPN,2,.OWNR,.CHKLOCK) then
00620                 ( SIGNALSEM(0) ; OUTERR(8) ) ;
00630                  ! no write access on  directory-created
00640             CURRENT = FINDSPACE(.ADDRESS);  ! look for free space
00650           if .CURRENT neq -1 then ( if .LINEAR[.CURRENT * 3] eql EOLIST
00660                 then   LINEAR[.CURRENT * 3  + 3] = EOLIST;  ! end of file
00670                 STORE = ENTERVAL(.CURRENT * 3);  ! create entry
00680                 if (.STORE eql -1 ) then
00690                       (SIGNALSEM(0);  OUTERR(14)  );
00700                 DISKIO(1,.TRACEBUF[STORAGE + .INDEX],LINEAR);
00710                 DISKIO(2,.ADDRESS,TRACEBUF) )
00720     else  begin
00730                 if .INDEX eql 54 then
00740                                 begin
00750                                   CURRENT = GETDESC(.NJOB);
00760                                    if (.CURRENT eql -1 )  then
00770                                       (  SIGNALSEM(0); OUTERR(14) ) ;
00780                                   PREVIOUS = .TRACEBUF[ CURDESC];
00790                                   TRACEBUF[ NEXTDESC] = .CURRENT;
00800                                   DISKIO(3,.PREVIOUS,TRACEBUF);
00810                                   INITDESC(.NJOB,.CURRENT,.PREVIOUS,TRACEBUF);
00820                                   INDEX = 0;
00830                                 end;
00840                 CURRENT = GETBLK(.NJOB);  ! allocate a new block
00850                 if ( .CURRENT eql -1 ) then (SIGNALSEM(0); OUTERR(15) );
00860                 DISKIO(0,.CURRENT ,LINEAR);
00870                  STORE  =  ENTERVAL(0); LINEAR[3] = EOLIST;
00880                  if ( .STORE eql -1 ) then (SIGNALSEM(0); OUTERR(14) );
00890                 ! create file entry in new block
00900                    DISKIO(1,.CURRENT,LINEAR);
00910                 TRACEBUF[ STORAGE + .INDEX ] = .CURRENT;
00920                  if (.INDEX lss 53 ) then
00930                       (TRACEBUF[STORAGE +.INDEX +1] = EOLIST  );
00940                 if .ADDRESS neq .TRACEBUF[ CURDESC] then
00950                                 begin
00960                                   DISKIO(3,.TRACEBUF[ CURDESC],TRACEBUF);
00970                                   DISKIO(2,.ADDRESS,TRACEBUF);
00980                                 end;
00990               TRACEBUF[ SIZE] = .TRACEBUF[ SIZE] + 1;  ! modify size
01000                 DISKIO(3,.ADDRESS,TRACEBUF);
01010           end;
01020             CURRENT = .OWNR;
01030           NEWDIR(.STORE,.PATH,.CHKLOCK,.CURRENT);  ! initializes descriptor
01040           end;
```

```
01050           SIGNALSEM(0);
01060         0
01070       end;
01080
01090
01100       end ELUDOM
```

```
00010     MODULE CREAT(NOLIST)=
00020     begin
00030
00040             %          AUTHOR      :  AARTI KUMAR
00050               DATE OF CREATION   :  MAY 25, 1982  %
00060
00070             % this is the file primitive used to creat text files
00080               it creates an entry in the appropriate file and
00090               then activates the file like in open      %
00100
00110
00120     require FILER1.B11;
00130     require FILER2.B11;
00140
00150     external GDATE,FINDSPACE,INITDESC,INITCHNL,FINDCHNL,ENTERVAL;
00160
00170     ROUTINE NEWDESC(ADDR,PATH,CHKLOCK,OWNR)=
00180             % this routine intializes a descriptor allocated to a
00190                new text file  %
00200
00210         begin
00220             DISKIO(2,.ADDR,TRACEBUF);
00230                     TRACEBUF[  FILENAME] =.(.PATH);
00240                     TRACEBUF[  FILENAME + 1] =.(.PATH +2);
00250                     TRACEBUF[  FILSTAT]<kind> = 0;
00260                     TRACEBUF[FILSTAT]<protection> = .CHKLOCK;
00270                     TRACEBUF[  SIZE] = 0;
00280                     TRACEBUF[PARENT ] = .ADDRESS;
00290                     TRACEBUF[  CURDESC] =.ADDR;
00300                     TRACEBUF[  NEXTDESC] = -1;
00310                     TRACEBUF[  OWNER] = .OWNR;
00320                     TRACEBUF[ DATE  ]  = GDATE();
00330                     TRACEBUF[  STORAGE] ='S';
00340             DISKIO(3,.ADDR,TRACEBUF);
00350         end;
00360
00370
00380     GLOBAL ROUTINE CREATOR(JOB, PPN,FILNAM,CHKLOCK,BUFPTR)=
00390
00400             % this is the file primitive "CREATOR" which is called to
00410               create text files    %
00420
00430       begin
00440         Local  TEMP,CURRENT,PREVIOUS;
00450
00460             WAITSEM(0);  ! check to see if resource associated with event 0 is free
00470             NJOB = .JOB;  FDERROR = 0;
00480             if .(.FILNAM) eql -1 then ADDRESS=ROOT else ADDRESS=.(.FILNAM);
00490             PATH=.FILNAM + 2;
00500             ADDRESS =TRACE(.ADDRESS,.PPN);
00510               ! this returns  address of the directory in which file exists
00520             if (.ADDRESS gtr ERRVAL  )  and (.ADDRESS lss MAXERR )
```

```
00530                   then  ( SIGNALSEM(0); RETURN .ADDRESS );
00540                         ! illegal path specification
00550         if not PROTCHK(.PPN,2,.TRACEBUF[OWNER],.TRACEBUF[FILSTAT])
00560                   then  ( SIGNALSEM(0); OUTERR(8) );  ! file protected against use
00570         CURRENT  = SEARCHDIREC(.ADDRESS);
00580         if ( .CURRENT neq -1  ) then  ( SIGNALSEM(0); OUTERR(3) );
00590                   ! file already exists
00600         DISKIO(2,.ADDRESS,TRACEBUF);
00610         if not PROTCHK(.PPN,2,.TRACEBUF[OWNER],.CHKLOCK) then
00620                   ( SIGNALSEM(0); OUTERR(8) );  ! file cannot be opened for write
00630           CURRENT = FINDSPACE(.ADDRESS);  ! look for free space in directory
00640         if .CURRENT neq -1 then ( if .LINEAR[.CURRENT * 3] eql '$'
00650                   then   LINEAR[.CURRENT * 3  + 3] = '$';
00660                 STORE = ENTERVAL(.CURRENT * 3);  ! create file entry
00670                 if (.STORE eql -1 ) then ( SIGNALSEM(0); OUTERR(14));
00680                 DISKIO(1,.TRACEBUF[ STORAGE + .INDEX],LINEAR);
00690                 DISKIO(2,.ADDRESS,TRACEBUF) )
00700   else  begin
00710                 if .INDEX eql 54 then
00720                                 begin
00730                                 ! allocate a new descriptor
00740                                   CURRENT = GETDESC(.NJOB);
00750                                 if (.CURRENT eql -1 ) then
00760                                   ( SIGNALSEM(0); OUTERR(15) );
00770                                   PREVIOUS = .TRACEBUF[ CURDESC];
00780                                   TRACEBUF[ NEXTDESC] = .CURRENT;
00790                                   DISKIO(3,.PREVIOUS,TRACEBUF);
00800                                   INITDESC(.NJOB,.CURRENT,.PREVIOUS,TRACEBUF);
00810                                   INDEX = 0;
00820                                 end;
00830                 CURRENT = GETBLK(.NJOB);  ! allocate a new block
00840                 if (.CURRENT eql -1 ) then ( SIGNALSEM(0); OUTERR(14));
00850                 DISKIO(0,.CURRENT ,LINEAR);
00860                  STORE  =   ENTERVAL(0); LINEAR[3] = '$';   ! create file entry
00870                 if ( .STORE eql -1 ) then ( SIGNALSEM(0); OUTERR(14) );
00880                    DISKIO(1,.CURRENT,LINEAR);
00890                 TRACEBUF[ STORAGE + .INDEX ] = .CURRENT;
00900                 if (.INDEX lss 53 ) then
00910                         (TRACEBUF[STORAGE +.INDEX +1] = '$'  );
00920                 if .ADDRESS neq .TRACEBUF[ CURDESC] then
00930                                 begin
00940                                   DISKIO(3,.TRACEBUF[ CURDESC],TRACEBUF);
00950                                   DISKIO(2,.ADDRESS,TRACEBUF);
00960                                 end;
00970             TRACEBUF[ SIZE] = .TRACEBUF[ SIZE] + 1;
00980                 ! modify size of directory file
00990               DISKIO(3,.ADDRESS,TRACEBUF);
01000         end;
01010         CURRENT =.TRACEBUF[ OWNER];
01020         NEWDESC(.STORE,.PATH,.CHKLOCK,.CURRENT);
01030         CHANNEL= FINDCHNL();
01040           if (.CHANNEL eql -1 ) then ( SIGNALSEM(0); OUTERR(9));
```

```
01050             Incr I to 63 do ( DESCRIPTOR[.CHANNEL,.I]_=.TRACEBUF[ .I] );
01060             INITCHNL(.CHANNEL,1,.NJOB,.BUFPTR);  ! initialize channel
01070             ACTIVE[.CHANNEL,FILSTAT1<extension> = .CHKLOCK<extension>;
01080             SIGNALSEM(0);  ! signal release of resource associated with event 0
01090       .CHANNEL        ! return the channel number
01100     end;
01110
01120
01130
01140     end ELJDUM
```

```
00010     MODULE OPENFILE(NOLIST)=
00020     begin
00030
00040             %       AUTHOR          :   AARTI KUMAR
00050               DATE OF CREATION    :   JUNE 1, 1982  %
00060
00070             % this program is the file primitive OPEN
00080               it takes as arguments PROCESS-NO, FILE-PATH-SPECIFICATION
00090               USER'S PPN , FUNCTION such as READ/READ-WRITE &  MEMORY
00100               LOCATION in user's area from where each diskio request
00110               is accomplished. %
00120
00130             %  A CHANNEL NUMBER IS RETURNED TO THE USER
00140               THIS MUST BE SPECIFIED FOR ANY FURTHER ACCESS ON
00150               THE OPEN FILE %
00160
00170
00180     require FILER1.B11;
00190     require FILER2.B11;
00200     external MATCH,INITCHNL,FINDCHNL;
00210
00220     GLOBAL ROUTINE OPEN(JOB, PPN,FILNAM,FUNCTION,BUFPTR)=
00230     begin
00240      local  VALUE,TEMP;
00250
00260       begin
00270             WAITSEM(0);       ! semaphore check for event 0
00280             NJOB = .JOB; FDERROR = 0;
00290             if .(.FILNAM) eql -1 then
00300                   ADDRESS=ROOT else ADDRESS=.(.FILNAM);
00310             PATH=.FILNAM + 2;
00320             ADDRESS =TRACE(.ADDRESS,.PPN);  ! 'trace' gets descriptor
00330                   !  address of directory in which file exists
00340             If (.ADDRESS gtr ERRVAL  )  and (.ADDRESS lss MAXERR )
00350                   then ( SIGNALSEM(0) ; RETURN .ADDRESS );
00360                     ! illegal path specification
00370             if not PROTCHK(.PPN,3,.TRACEBUF[OWNER],.TRACEBUF[FILSTAT] )
00380                   then (SIGNALSEM(0); OUTERR(8)  );
00390                  ! check for read access on directory
00400             ADDRESS = SEARCHDIREC(.ADDRESS);  ! search for file in directory
00410             if .ADDRESS eql -1 then (SIGNALSEM(0); OUTERR(2));
00420             !  error = file not found
00430             DISKIO(2,.ADDRESS,TRACEBUF);
00440             if .FUNCTION then TEMP = 2 else TEMP = 3 ;
00450             if not PROTCHK(.PPN,.TEMP,.TRACEBUF[OWNER],.TRACEBUF[FILSTAT] )
00460                   then (SIGNALSEM(0); OUTERR(8) );
00470                              ! file protected against use
00480             If .TRACEBUF[ FILSTAT]<kind> then  (if .FUNCTION
00490                              then (SIGNALSEM(0); OUTERR(5)) )
00500                   ! directory file cannot be opened for write
00510                 else
00520                   begin
```

```
00530                       if .FUNCTION then (if MATCH(.ADDRESS,2)
00540                             then  (SIGNALSEM(0); OUTERR(6)))
00550                           ! file opened earlier
00560                           else (if MATCH(.ADDRESS,1)
00570                                 then (SIGNALSEM(0); OUTERR(7)) )
00580                     ! file opened for write earlier
00590                     end;
00600          CHANNEL = FINDCHNL();
00610       if .CHANNEL eql -1 then (SIGNALSEM(0); OUTERR (9) );
00620          ! no free channel available
00630          incr I to 63 do (DESCRIPTOR[.CHANNEL,.I] = .TRACEBUF[ .I] );
00640          !  transfer file descriptor in specified channel
00650          INITCHNL(.CHANNEL,.FUNCTION,.JOB,.BUFPTR);
00660          ACTIVE[.CHANNEL,FILSTAT]<extension> = .FUNCTION<extension>;
00670          SIGNALSEM(0);    ! signal to semaphore for release of resource 0
00680        .CHANNEL
00690    end
00700 end;
00710
00720
00730 end  ELUDOM
```

```
00010     MODULE GETFIL(NOLIST)=
00020      begin
00030
00040             % file primitive to fetch a file block  %
00050
00060             %       author       :  AARTI  KUMAR
00070               date of creation :  1 JUNE , 1982      %
00080
00090     require FILER1.B11;
00100     EXTERNAL DIVL,MODL;
00110     EXTERNAL WRTDESC,FETCHDESC;
00120
00130
00140     GLOBAL ROUTINE GET(JOB,CH) =
00150       begin
00160         Local LOCN,FUNC,DESCNUM,BLOCK,OFFSET,POINTER,CHNL,CURNUMB;
00170
00180             % routine to get file blocks  %
00190
00200             CHNL = .CH;
00210             if .JOB neq .ACTIVE[.CHNL,JOBNO] then ABORT(.JOB,10);
00220             if .ACTIVE[.CHNL,CURBLK] eql .ACTIVE[.CHNL,SIZE] then ABORT(.JOB,12);
00230             DESCNUM = DIVL( .ACTIVE[.CHNL,CURBLK], 54);
00240             OFFSET  = MODL( .ACTIVE[.CHNL,CURBLK] ,54);
00250             CURNUMB = .ACTIVE[.CHNL,FILSTAT]<descno>;
00260             if .DESCNUM neq .CURNUMB then
00270                     begin
00280                         if  .ACTIVE[.CHNL,FILSTAT]<curmod> then WRTDESC(.CHNL);
00290                          ACTIVE[.CHNL,FILSTAT]<curmod> = 0;
00300                          FETCHDESC(.DESCNUM,CURNUMB,.CHNL);
00310                      end;
00320             BLOCK = .DESCRIPTOR[.CHNL, STORAGE + .OFFSET];
00330             LOCN = .ACTIVE[.CHNL,BUFADDR];
00340             FUNC<extension> = .ACTIVE[.CHNL,FILSTAT]<extension>;
00350             FUNC<oprn> = 0;
00360             ENTEREQUEST(.JOB,.FUNC,.BLOCK,.LOCN);
00370             ACTIVE[.CHNL,CURBLK] = .ACTIVE[.CHNL,CURBLK] + 1;
00380             if .ACTIVE[.CHNL,CURBLK] eql .ACTIVE[.CHNL,SIZE] then OUTMSG(-1);
00390          .ACTIVE[.CHNL,CURBLK]
00400       end;
00410
00420
00430
00440     end ELUDOM
```

```
00010     MODULE PUTFIL(NOLIST)=
00020     begin
00030
00040               %  file primitive to put blocks in a file %
00050
00060               %          author          :  AARTI  KUMAR
00070                 date of creation   :  20 MAY , 1982    %
00080
00090     require FILER1.R11;
00100     EXTERNAL DIVL,MODL;
00110     EXTERNAL WRTDESC,FETCHDESC;
00120     external INITDESC;
00130
00140
00150     GLOBAL ROUTINE PUT(JOB ,CH) =
00160       begin
00170         Local FUNC,MLOCN,FBLK,CHNL,OFFSET,DESCNUM,XADDR,YADDR,CURNUMB;
00180
00190               %  routine to put blocks in a file  %
00200
00210               CHNL = .CH;
00220               if .JOB neq .ACTIVE[.CHNL,JOBNO] then ABORT(.JOB,10);
00230               if not .ACTIVE[.CHNL,FILSTAT]<wrtbit>  then ABORT(.JOB,11);
00240               DESCNUM = DIVL(.ACTIVE[.CHNL,CURBLK],54);
00250               OFFSET  = MODL(.ACTIVE[.CHNL,CURBLK],54);
00260               CURNUMB = .ACTIVE[.CHNL,FILSTAT]<descno>;
00270               if .CURNUMB neq .DESCNUM then
00280                       begin
00290                           if .ACTIVE[.CHNL,FILSTAT]<curmod> then WRTDESC(.CHNL);
00300                            ACTIVE[.CHNL,FILSTAT]<curmod> = 0;
00310                            FETCHDESC(.DESCNUM,CURNUMB,.CHNL);
00320                       end;
00330               if .CURNUMB neq .DESCNUM then
00340                       begin
00350                               XADDR = GETDESC(.JOB);
00360                               if .XADDR eql -1 then ABORT(.JOB,14);
00370                               YADDR = .DESCRIPTOR[.CHNL,CURDESC];
00380                               DESCRIPTOR[.CHNL, NEXTDESC] = .XADDR;
00390                               WRTDESC(.CHNL);
00400                               INITDESC(.JOB,.XADDR,.YADDR,DESCRIPTOR[.CHNL,0]);
00410                               CURNUMB = .CURNUMB + 1;
00420                               ACTIVE[.CHNL,FILSTAT]<descno> =.CURNUMB;
00430                       end;
00440               if .ACTIVE[.CHNL,CURBLK] eql .ACTIVE[.CHNL,SIZE]
00450                     then  (   FBLK = GETBLK(.JOB);
00460                               if .FBLK eql -1 then ABORT(.JOB,15)   )
00470                     else FBLK = .DESCRIPTOR[.CHNL,STORAGE +.OFFSET];
00480               MLOCN = .ACTIVE[.CHNL,BUFADDR];
00490               FUNC<oprn> = 1;
00500               FUNC<extension> = .ACTIVE[.CHNL,FILSTAT]<extension>;
00510               ENTEREQUEST(.JOB,.FUNC,.FBLK,.MLOCN);
00520               if .ACTIVE[.CHNL,CURBLK] eql .ACTIVE[.CHNL,SIZE] then
```

```
00530                       begin
00540                             DESCRIPTOR[.CHNL,STORAGE + .OFFSET] = .FBLK;
00550                             if (.OFFSET ) lss 53 then
00560                             DESCRIPTOR[.CHNL,STORAGE + .OFFSET + 1] = EOLIST;
00570                             ACTIVE[.CHNL,FILSTAT]<curmod> =1;
00580                             ACTIVE[.CHNL,FILSTAT]<firstmod> =1;
00590                             ACTIVE[.CHNL,SIZE] =.ACTIVE[.CHNL,SIZE] + 1;
00600                       end;
00610               ACTIVE[.CHNL,CURBLK] =.ACTIVE[.CHNL,CURBLK] + 1;
00620               if .ACTIVE[.CHNL,CURBLK] eql .ACTIVE[.CHNL,SIZE] then OUTMSG(-1);
00630         .ACTIVE[.CHNL,CURBLK]
00640       end;
00650
00660
00670       end ELUDOM
```

```
00010     MODULE PUTMRKEOF(NOLIST)=
00020      begin
00030
00040             %  puts an end of file mark in the storage map  %
00050
00060             %      author        :  AARTI  KUMAR
00070              date of creation  :  5 JUNE, 1982     %
00080
00090     require FILER1.B11;
00100     EXTERNAL DIVL,MODL;
00110     EXTERNAL WRTDESC,FETCHDESC;
00120
00130     GLOBAL ROUTINE PUTEOF(JOB,CHNL) =
00140     begin
00150        Local DESCNUM,ADDR,COUNT,OFFSET,PTR,CURNUMB;
00160
00170             % routine called to put an end of file mark %
00180
00190
00200             if .ACTIVE[.CHNL,JOBNO] neq .JOB then ABORT(.JOB,10);
00210             if not .ACTIVE[.CHNL,FILSTAT]<wrtbit> then ABORT(.JOB,11);
00220             if .ACTIVE[.CHNL,CURBLK] eql .ACTIVE[.CHNL,SIZE] then RETURN 0;
00230             DESCNUM = DIVL(.ACTIVE[.CHNL,CURBLK] ,54);
00240             OFFSET = MODL(.ACTIVE[.CHNL,CURBLK] ,54);
00250             CURNUMB = .ACTIVE[.CHNL,FILSTAT]<descno> ;
00260             if .DESCNUM neq .CURNUMB then
00270                     begin
00280                             if .ACTIVE[.CHNL,FILSTAT]<curmod> then  WRTDESC(.CHNL);
00290                             ACTIVE[.CHNL,FILSTAT]<curmod> = 0;
00300                             FETCHDESC(.DESCNUM,CURNUMB,.CHNL);
00310                     end;
00320             COUNT = 0;
00330             Incr I from 0 to ( 53 - .OFFSET) do
00340                     begin
00350                         if .DESCRIPTOR[.CHNL,STORAGE + .OFFSET + .I ] eql EOLIST
00360                             then EXITLOOP;
00370                             PUTBLK(.JOB , .DESCRIPTOR[.CHNL,STORAGE + .OFFSET + .I ] );
00380                             COUNT = .COUNT + 1 ;
00390                     end;
00400             if (.OFFSET eql 0 ) and (.CURNUMB gtr 0 )
00410                             then begin
00420                                     ADDR = .DESCRIPTOR[.CHNL,PREVDESC] ;
00430                                     PUTDESC(.JOB,.DESCRIPTOR[.CHNL,CURDESC] );
00440                                     ENTEREQUEST(.JOB,  2,.ADDR,DESCRIPTOR[.CHNL,0] );
00450                                     DESCRIPTOR[.CHNL,NEXTDESC] = -1;
00460                                     ACTIVE[.CHNL,FILSTAT] = .CURNUMB - 1;
00470                                     if .ACTIVE[.CHNL,FILSTAT]<descno> neq 0
00480                                     then WRTDESC(.CHNL);
00490                             end
00500           else  begin
00510                     DESCRIPTOR[.CHNL,STORAGE + .OFFSET ] = EOLIST;
00520                     if .COUNT eql ( 54 - .OFFSET) then
```

```
00530                                   begin
00540                                    ADDR = .DESCRIPTOR[.CHNL,NEXTDESC];
00550                                    if  .ADDR neq -1 then
00560                                       begin
00570                                           DESCRIPTOR[.CHNL,NEXTDESC] = -1;
00580                                           WRTDESC(.CHNL);
00590                                           PTR = .DESCRIPTOR[.CHNL,CURDESC];
00600                                           do
00610                                             begin
00620                                               ENTEREQUEST(.JOB, 2 ,.ADDR,DESCRIPTOR[.CHNL,0]);
00630                                               Incr I from 0 to 53 do
00640                                               (if .DESCRIPTOR[.CHNL, STORAGE + .I ]
00650                                                     eql EOLIST then EXITLOOP ;
00660                                                  PUTBLK(.JOB,.DESCRIPTOR[.CHNL, STORAGE + .I] );
00670                                                    COUNT = .COUNT + 1   );
00680                                                    PUTDESC(.JOB,.ADDR);
00690                                                    ADDR = .DESCRIPTOR[.CHNL, NEXTDESC];
00700                                             end
00710                                           until (.ADDR eql -1);
00720                                           ENTEREQUEST(.JOB, 2,.PTR,DESCRIPTOR[.CHNL,0]);
00730                                       end;
00740                                   end;
00750             end;
00760             ACTIVE[.CHNL,SIZE] = .ACTIVE[.CHNL,SIZE] - .COUNT ;
00770             ACTIVE[.CHNL,FILSTAT]<firstmod> = 1;
00780             ACTIVE[.CHNL,FILSTAT]<curmod> = 0;
00790         .COUNT
00800     end;
00810
00820
00830     end ELUDOM
```

```
00010     MODULE SEEKBLK(NOLIST)=
00020     begin
00030             % file primitive to set the current block pointer  %
00040
00050             %       author       :  AARTI  KUMAR
00060               date of creation : 22 MAY , 1982     %
00070
00080     require FILER1.B11;
00090
00100     GLOBAL ROUTINE SEEK(JOB,CHNL,FBLK) =
00110     begin
00120             % current block pointer is set at value specified %
00130
00140             If .JOB neq .ACTIVE[.CHNL,JOBNO] then ABORT(.JOB,10);
00150             If .FBLK geq .ACTIVE[.CHNL,SIZE] then
00160                               begin
00170                                  ACTIVE[.CHNL,CURBLK] = .ACTIVE[.CHNL,SIZE] ;
00180                                       OUTMSG(-1); !message to indicate end of file
00190                               end
00200             else ACTIVE[.CHNL,CURBLK] = .FBLK;
00210         .FBLK
00220     end;
00230
00240
00250     end ELUDOM
```

```
00010     MODULE BLKPTRCHK(NOLIST)=
00020     begin
00030
00040             % file primitive to check  the current block pointer  %
00050
00060             %       author          :  AARTI  KUMAR
00070               date of creation : 22 MAY , 1982     %
00080
00090      require FILER1.b11;
00100
00110     GLOBAL ROUTINE BLKCHK(JOB,CHNL)=
00120       begin
00130             % returns the current block pointer value  %
00140
00150             If .JOB neq .ACTIVE[.CHNL,JOBNO] then ABORT(.JOB,10);
00160             if .ACTIVE[.CHNL,CURBLK] eql .ACTIVE[.CHNL,SIZE]
00170                then  OUTMSG(-1) ;   ! end of file indicated
00180             .ACTIVE[.CHNL,CURBLK]
00190       end;
00200
00210     end ELUDOM
```

```
00010     MODULE CLOSEFILE(STACK,NOLIST)=
00020     begin
00030             % file primitive to deactivate files  %
00040
00050             %       author       :  AARTI  KUMAR
00060               date of creation : 25 MAY , 1982     %
00070
00080     require FILER1.R11;
00090
00100     external WRTDESC;
00110
00120     GLOBAL ROUTINE CLOSE(JOB,CH) =
00130     begin
00140       Local POINTER,CHNL,ADDR,CURNT;
00150
00160             % routine to close files %
00170
00180             CHNL = .CH;
00190             if .ACTIVE[.CHNL,JOBNO] neq .JOB then  ABORT(.JOB,10);
00200              if .ACTIVE[.CHNL,FILSTAT]<delbits> eql 1 then
00210                 begin
00220                     if .ACTIVE[.CHNL,FILSTAT]<descno> neq 0 then
00230                     (ENTEREQUEST(.JOB,2,.ACTIVE[.CHNL,PARENT] ,DESCRIPTOR[.CHNL,0]));
00240                     do begin
00250                             Incr I from 0 to 53 do
00260                             begin
00270                                if .DESCRIPTOR[.CHNL,STORAGE + .I] eql EOLIST
00280                                  then EXITLOOP else
00290                             (PUTBLK(.JOB,.DESCRIPTOR[.CHNL,STORAGE + .I]) );
00300                             end;
00310                             CURNT = .DESCRIPTOR[.CHNL,NEXTDESC];
00320                             PUTDESC(.JOB,.DESCRIPTOR[.CHNL,CURDESC]);
00330                          if .CURNT neq -1 then
00340                          ( ENTEREQUEST(.JOB,2,.CURNT,DESCRIPTOR[.CHNL,0]) );
00350                        end
00360                        until .CURNT eql -1;
00370                end
00380                 else
00390                     if .ACTIVE[.CHNL,FILSTAT]<delbits> gtr 1 then
00400                     begin
00410                             Incr I from 0 to (MAXM-1) do
00420                             (if .ACTIVE[.CHNL,PARENT] eql   .ACTIVE[.I,PARENT] then
00430                             (ACTIVE[.I,FILSTAT]<delbits> =
00440                                     .ACTIVE[.CHNL,PARENT]<delbits> - 1 ) );
00450                     end
00460        else begin
00470              if ( .ACTIVE[.CHNL,FILSTAT]<curmod> and
00480                 ( .ACTIVE[.CHNL,FILSTAT]<descno> neq 0 ) )   then
00490                             begin
00500                                WRTDESC(.CHNL);
00510                                ACTIVE[.CHNL,FILSTAT]<curmod> = 0;
00520                                 if .ACTIVE[.CHNL,FILSTAT]<firstmod>
```

```
00530                                   then    ( ADDR  = .ACTIVE[.CHNL,PARENT];
00540                                     ENTEREQUEST(.JOB,2,.ADDR,DESCRIPTOR[.CHNL,01);
00550                                   ACTIVE[.CHNL,FILSTAT]<descno> = 0 );
00560                                end;
00570              if .ACTIVE[.CHNL,FILSTAT]<firstmod> then
00580                                begin
00590                                   DESCRIPTOR[.CHNL,SIZE] = .ACTIVE[.CHNL,SIZE] ;
00600                                  WPIDESC(.CHNL);
00610                                end;
00620              end;
00630              ACTIVE[.CHNL,JOBNO] = -1;
00640              Incr I from 1 to 5 do (ACTIVE[.CHNL,.I] = 0 );
00650          0
00660      end;
00670
00680
00690      end ELUDOM
```

```
00010     MODULE CURDIRPIP(NOLIST)=
00020     begin
00030             % file primitive to set the current directory pointer  %
00040
00050             %       author       :  AARTI  KUMAR
00060               date of creation : 10 JUNE, 1982   %
00070
00080      require FILER1.B11;
00090      require FILER2.B11;
00100
00110     GLOBAL ROUTINE CDPCHK( JOB,PPN,FILNAM) =
00120     begin
00130             % this routine returns descriptor address of directory %
00140
00150             WAITSEM(0);
00160             MJOB = .JOB;  FDERROR = 0;
00170             PATH = .FILNAM; ADDRESS = ROOT;
00180             ADDRESS = TRACE(.ADDRESS,.PPN);
00190             if ( .ADDRESS gtr ERRVAL) and (.ADDRESS lss MAXERR) then
00200             ( SIGNALSEM(0); RETURN  .ADDRESS );
00210             if not PROTCHK(.PPN,3,.TRACEBUF[OWNER],.TRACEBUF[FILSTAT] )
00220              then  ( SIGNALSEM(0); OUTERR(8) ) ;
00230             ADDRESS = SEARCHDIREC(.ADDRESS);
00240             if .ADDRESS eql -1 then  ( SIGNALSEM(0); OUTERR(2) );
00250             DISKIO(2,.ADDRESS,TRACEBUF);
00260             if not .(TRACEBUF[FILSTAT]<kind>) then  ( SIGNALSEM(0) ; OUTERR(4));
00270             SIGNALSEM(0);
00280             .ADDRESS
00290     end;
00300
00310
00320     end ELUDOM
```

```
00010     MODULE RENAMFILE(NOLIST)=
00020     begin
00030             % file primitive to rename files  and change protection  %
00040
00050             %       author       :  AARTI  KUMAR
00060               date of creation : 15 JUNE , 1982     %
00070
00080     require FILER1.B11;
00090     require FILER2.B11;
00100
00110     GLOBAL ROUTINE RENAME(JOB,PPN,FILNAM,NEWLOCK)=
00120     begin
00130       local FLAG,J,CURRENT,PTEMP,STEMP;
00140
00150             % this routine is called to rename or change protection
00160                for a file   %
00170
00180             WAITSEM(0);
00190             NJOB= .JOB; FDERROR = 0;
00200             If .(.FILNAM) eql -1 then ADDRESS = ROOT else ADDRESS = .(.FILNAM) ;
00210             PATH = .FILNAM + 2;
00220             ADDRESS = TRACE(.ADDRESS,.PPN) ;
00230             If (.ADDRESS gtr ERRVAL )  and (.ADDRESS lss MAXERR)
00240              then (  SIGNALSEM(0) ; RETURN .ADDRESS );
00250             If not PROTCHK(.PPN,2,.TRACEBUF[OWNER] ,.TRACEBUF[FILSTAT] )
00260              then ( SIGNALSEM(0); OUTERR(8) );
00270             If ( .(.PATH + 6) neq -1   ) then
00280                     begin
00290                             PTEMP = .PATH;
00300                             PATH =.PATH + 4;
00310                             (.PATH ) =.ADDRESS;
00320                             (.PATH + 6 ) =EOLIST;
00330                             STEMP = SEARCHDIREC(.ADDRESS);
00340                             IF .STEMP NEQ -1 then ( SIGNALSEM(0); OUTERR(1) );
00350                             PATH =.PTEMP;
00360                             (.PATH + 4) = EOLIST;
00370                     end;
00380             CURRENT = SEARCHDIREC(.ADDRESS);
00390             if .CURRENT eql -1 then ( SIGNALSEM(0); OUTERR(2) );
00400             DISKIO(2,.CURRENT,TRACEBUF);
00410             If not PROTCHK(.PPN,1,.TRACEBUF[OWNER],.TRACEBUF[FILSTAT] )
00420             then ( SIGNALSEM(0); OUTERR(8) ) ;    ! access to rename prohibited
00430             If ( .(.PATH + 6) neq -1   ) then
00440                 begin
00450                     FLAG = 0; J = 0;
00460                     do begin
00470                       if (( .LINEAR[.J*3] eql .(.PATH)) and
00480                           (.LINEAR[.J*3+1] eql .(.PATH+2)) )
00490                         then FLAG = 1 else J = .J+1;
00500                     end
00510                     until ( .J eql MAXM) or ( .FLAG);
00520                         PATH = .PATH + 6;
```

```
00530                       LINEAR[.J * 3] = .(.PATH);
00540                        LINEAR[.J * 3 + 1] = .(.PATH + 2);
00550                       DISKIO(1,.STORE,LINEAR);
00560                        TRACEBUF[FILENAME] = .(.PATH);
00570                        TRACEBUF[FILENAME + 1] = .(.PATH + 2);
00580                  end;
00590            If .NEWLOCK  neq -1 then
00600                  begin
00610                     if .PPN neq .TRACEBUF[OWNER] then
00620                  ( if not PROTCHK(.PPN,0,.TRACEBUF[OWNER],.TRACEBUF[FILSTAT] )
00630                                 then ( SIGNALSEM(0); OUTERR(8) )  );
00640                     TRACEBUF[FILSTAT]<protection> = .NEWLOCK
00650                  end;
00660            DISKIO(3,.CURRENT,TRACEBUF);
00670            Incr I from 0 to (MAXM -1 ) do
00680                  begin
00690                    if .ACTIVE[.I,PARENT] eql .CURRENT then
00700                       begin
00710                         If .ACTIVE[.I,FILSTAT]<descno> eql 0
00720                         then (DESCRIPTOR[.I,FILENAME] = .TRACEBUF[FILENAME];
00730                           DESCRIPTOR[.I,FILENAME + 1] = .TRACEBUF[FILENAME + 1];
00740                           DESCRIPTOR[.I,FILSTAT] = .TRACEBUF[FILSTAT] );
00750                       end;
00760                  end;
00770            SIGNALSEM(0);
00780            0
00790      end;
00800
00810
00820
00830      END ELUDOM
```

```
00010     MODULE DELETFILE(NOLIST)=
00020     begin
00030             % file primitive delete files  %
00040
00050             %       author       :  AARTI  KUMAR
00060               date of creation : 22 JUNE , 1982     %
00070
00080     require FILER1.R11;
00090     require FILER2.R11;
00100
00110     external MATCH;
00120
00130     ROUTINE FREENTRY(POSITION)=
00140     begin
00150         Local DELSIZE,TEMP,FLAG,I;
00160
00170             % this routine frees the file entry from the directory %
00180
00190
00200          If (.LINEAR[.POSITION * 3 + 3 ] eql EOLIST) or (.POSITION eql 41)
00210             then begin
00220                     If (.POSITION eql 41 )
00230                       then (If .TRACEBUF[STORAGE + .INDEX + 1] neq EOLIST
00240                             then begin
00250                                     LINEAR[.POSITION * 3] = -1;
00260                                     LINEAR[.POSITION * 3 + 1] = 0;
00270                                     LINEAR[.POSITION * 3 + 2] = 0;
00280                                     DISKIO(1,.STORE,LINEAR);
00290                                     RETURN
00300                                   end ) ;
00310                   DELSIZE = FLAG = 0; I = 1;
00320                While not .FLAG do
00330             begin
00340                While (.POSITION - .I) geq 0 do
00350                             begin
00360                                If .LINEAR[(.POSITION - .I) * 3] neq -1 then
00370                                  (Linear[(.POSITION - .I)*3 +3] = EOLIST;
00380                                                DISKIO(1,.STORE,LINEAR);
00390                                                FLAG = 1; EXITLOOP)
00400                                     else I = .I + 1;
00410                             end;
00420                    If (.POSITION - .I) lss 0 then
00430                             begin
00440                                PUTBLK(.NJOB,.STORE);
00450                                DELSIZE = .DELSIZE + 1;
00460                                TRACEBUF[STORAGE + .INDEX] = EOLIST ;
00470                                        If .INDEX neq 0 then
00480                                                (INDEX = .INDEX - 1;
00490                                                STORE = .TRACEBUF[STORAGE + .INDEX] ;
00500                                                DISKIO(0,.STORE,LINEAR);
00510                                                  POSITION = 41;
00520                                                  I = 0 ;FLAG = 0 )
```

```
00530                                       else begin
00540                                            If .ADDRESS neq .TRACEBUF[CURDESC] then
00550                                            (TEMP = .TRACEBUF[PREVDESC];
00560                                              PUTDESC(.NJOB,.TRACEBUF[CURDESC]);
00570                                               DISKIO(2,.TEMP,TRACEBUF);
00580                                                    INDEX = 53;
00590                                               POSITION = 41; I = 0; FLAG = 0)
00600                                            else (FLAG = 1 );
00610                                       end;
00620                          end;
00630                  end;
00640          If .TRACEBUF[CURDESC] neq .ADDRESS then
00650                                  begin
00660                                    DISKIO(3,.TRACEBUF[CURDESC],TRACEBUF);
00670                                    DISKIO(2,.ADDRESS,TRACEBUF);
00680                                  end;
00690               TRACEBUF[SIZE] = .TRACEBUF[SIZE] - .DELSIZE;
00700                  DISKIO(3,.ADDRESS,TRACEBUF);
00710          end
00720          else begin
00730                   LINEAR[.POSITION * 3] = -1;
00740                   LINEAR[.POSITION * 3 + 1] = 0;
00750                   LINEAR[.POSITION * 3 + 2] = 0;
00760                            DISKIO(1,.STORE,LINEAR);
00770             end;
00780  end;
00790
00800
00810
00820  GLOBAL ROUTINE DELETE(JOB,PPN,FILNAM)=
00830  begin
00840   Local POS,COUNT,I,J,CURRENT;
00850
00860          % this routine is called to delete files %
00870
00880          WAITSEM(0);
00890          NJOB = .JOB;  FDERROR = 0 ;
00900      If .(.FILNAM) eql -1 then ADDRESS = ROOT else ADDRESS = .(.FILNAM);
00910          PATH = .FILNAM + 2;
00920          ADDRESS = TRACE(.ADDRESS,.PPN);
00930          If (.ADDRESS gtr ERRVAL) and (.ADDRESS lss MAXERR) then
00940          (SIGNALSEM(0); RETURN  .ADDRESS );
00950          If not PROTCHK(.PPN,2,.TRACEBUF[OWNER],.TRACEBUF[FILSTAT])
00960           then  (SIGNALSEM(0); OUTERR(8) );
00970          CURRENT = SEARCHDIREC(.ADDRESS);
00980          If .CURRENT eql -1 then  ( SIGNALSEM(0); OUTERR(2) );
00990             DISKIO(2,.CURRENT,TRACEBUF);
01000          If not PROTCHK(.PPN,1,.TRACEBUF[OWNER],.TRACEBUF[FILSTAT] )
01010           then  ( SIGNALSEM(0); OUTERR(8) );
01020          If .TRACEBUF[FILSTAT]<kind> then
01030          (if .TRACEBUF[SIZE] gtr 0 then  (SIGNALSEM(0); OUTERR(16) ));
01040          if MATCH(.CURRENT,2) then
```

```
01050                                 begin I = 0; J = 0;
01060                                while .I lss MAXM do
01070                                 (if .ACTIVE[.I,PARENT] eql .CURRENT
01080                                  then (J = .J + 1); I = .I + 1);
01090                                 Incr K from 0 to (MAXM - 1) do
01100                                  (if .ACTIVE[.K,PARENT] eql .CURRENT then
01110                                     ACTIVE[.K,FILSTAT]<delbits> = .J );
01120                                        COUNT = .TRACEBUF[SIZE] ;
01130                                 end
01140          else begin
01150               COUNT = 0;
01160            Do
01170               begin
01180                  DISKIO(2,.CURRENT,TRACEBUF);
01190                  Incr I from 0 to 53 do
01200                                 begin
01210                                   if .TRACEBUF[STORAGE+.I] eql EOLIST
01220                                     then EXITLOOP else
01230                                       (PUTBLK(.NJOB,.TRACEBUF[ STORAGE+.I]);
01240                                        COUNT = .COUNT + 1);
01250                                 end;
01260                  CURRENT = .TRACEBUF[NEXTDESC];
01270                  PUTDESC(.NJOB,.TRACEBUF[CURDESC]);
01280               end
01290            until .CURRENT eql -1;
01300             end;
01310               DISKIO(2,.ADDRESS,TRACEBUF);
01320               CURRENT = SEARCHDIREC(.ADDRESS);
01330               POS = (  Incr I from 0 to 41 do ( if ( (.LINEAR[.I* 3] eql .(.PATH) )
01340                                 and ( .LINEAR[.I*3+1] eql .(.PATH +2) ) )
01350                                   then EXITLOOP .I) );
01360               FREENTRY(.POS);
01370               SIGNALSEM(0);
01380          .COUNT
01390     end;
01400
01410
01420     END ELUDOM
```

```
00010     MODULE  ROUTNES(NOLIST) =
00020     begin
00030             %  common routines for the filing system %
00040
00050             %       author       :  AARTI  KUMAR
00060               date of creation : 10 MAY , 1982     %
00070
00080
00090     require FILFR1.R11;
00100     external TRACEBUF,LINEAR;
00110     map VECTOR TRACEBUF ,VECTOR LINEAR;
00120
00130     external STORE,INDEX,PATH,ADDRESS,CHANNEL,NJOB,FDERROR;
00140     external DISKIO;
00150
00160
00170     MACRO OUTERR(NUMB) =
00180             ( FDERROR = .NUMB + ERRVAL ; RETURN -1 )%;
00190
00200
00210     GLOBAL ROUTINE FETCHDESC(NUMBER,CURNUMB,CHNL)=
00220     begin
00230       local  BOOL, FBLK;
00240
00250             % this routine fetches the right descriptor in memory %
00260
00270      BOOL = 0;
00280       do
00290         begin
00300             if .(.CURNUMB ) GTR .NUMBER then ( (.CURNUMB) = .(.CURNUMB )  - 1;
00310                                     FBLK = .DESCRIPTOR[.CHNL, PREVDESC] )
00320                                  else ( (.CURNUMB ) = .(.CURNUMB ) + 1;
00330                                     FBLK = .DESCRIPTOR[.CHNL, NEXTDESC] );
00340             if .FBLK eql -1 then ( (.CURNUMB )= .(.CURNUMB )-1;
00350                                     BOOL = 1;
00360                                     FBLK = .DESCRIPTOR[.CHNL, CURDESC] )
00370             else    ( ENTEREQUEST(.ACTIVE[.CHNL,JOBNO],2,.FBLK,DESCRIPTOR[.CHNL,0]) );
00380         end
00390       until ( .(.CURNUMB ) eql .NUMBER ) OR .BOOL;
00400             ACTIVE[.CHNL,FILSTAT]<descno> = .(.CURNUMB );
00410     end;
00420
00430
00440     GLOBAL ROUTINE WRTDESC(CHNL) =
00450     begin
00460      local ADDR;
00470             % this routine writes back the descriptor from a specified
00480                channel   %
00490
00500
00510             ADDR = .DESCRIPTOR[.CHNL,CURDESC];
00520             ENTEREQUEST(.ACTIVE[.CHNL,JOBNO],3,.ADDR,DESCRIPTOR[.CHNL,0]);
```

```
00530     end;
00540
00550
00560     GLOBAL  ROUTINE ENTERVAL(POSITION) =
00570     begin
00580             % this enters a directory entry %
00590
00600             LINEAR[.POSITION] = .(.PATH);
00610             LINEAR[.POSITION +1] =.(.PATH +2);
00620             LINEAR[.POSITION + 2] = GETDESC(.NJOB);
00630           .LINEAR[.POSITION + 2]
00640     end;
00650
00660
00670     GLOBAL ROUTINE FINDSPACE(ADDRESS) =
00680     begin
00690
00700             % this finds a free space in the directory %
00710
00720
00730             STORE=INDEX=0;
00740             STORE = .TRACEBUF[ STORAGE];
00750               While (.STORE neq EOLIST ) do
00760                     begin
00770                         DISKIO(0,.STORE,LINEAR);
00780                          (Incr I to 41 do
00790                               (if (.LINEAR[.I * 3 ] eql  -1 )  or
00800                                     (.LINEAR[.I * 3] eql EOLIST)
00810                                           then RETURN .I));
00820                            INDEX = .INDEX + 1;
00830                            if (.INDEX eql 54) then (ADDRESS = .TRACEBUF[ NEXTDESC];
00840                              if .ADDRESS eql -1 then RETURN -1;
00850                            DISKIO(2,.ADDRESS,TRACEBUF);
00860                               INDEX = 0 );
00870                               STORE = .TRACEBUF[ STORAGE + .INDEX];
00880                     end;
00890             -1
00900     end;
00910
00920      GLOBAL  ROUTINE INITDESC(JOB,CURRENT,PREVIOUS,BUFFER) =
00930     begin
00940
00950             % initializes second and later descriptors %
00960
00970
00980           ENTEREQUEST(.JOB,2,.CURRENT,.BUFFER);
00990             (.BUFFER)[ PREVDESC] = .PREVIOUS;
01000             (.BUFFER)[ CURDESC] = .CURRENT;
01010             (.BUFFER)[ NEXTDESC] = -1;
01020           ENTEREQUEST(.JOB,3,.CURRENT,.BUFFER);
01030     end;
01040
```

```
01050
01060     GLOBAL  ROUTINE MATCH(DSKADD,FUNC) =
01070     begin
01080     local   BOOL,I;
01090
01100             % this routine looks for a match of file in active table %
01110
01120                     BOOL = 0;  I =0;
01130             while (not .BOOL )  and (.I lss MAXM )     do
01140             begin
01150                      if .ACTIVE[.I,PARENT ] eql .DSKADD then
01160                        select .FUNC OF
01170                     NSET
01180                        0:if  not  .ACTIVE[.I,FILSTAT]<wrtbit> then BOOL = 1;
01190                        1:if .ACTIVE[.I,FILSTAT]<wrtbit> then BOOL = 1;
01200                   otherwise: BOOL = 1;
01210                     TES ;
01220                     I =.I +1;
01230             end;
01240        .BOOL
01250     end;
01260
01270
01280     GLOBAL ROUTINE SEARCHDIREC(FILADDR) =
01290     begin
01300
01310             % this searches for a file in a directory %
01320
01330             STORE =INDEX=0;
01340             STORE = .TRACEBUF[ STORAGE ];
01350               while (.STORE neq EOLIST ) do
01360                     begin
01370                       DISKIO(0,.STORE,LINEAR);
01380                     ( Incr I to 41 do
01390                             (if .LINEAR[.I*3] eql .(.PATH)
01400                                     and .LINEAR[.I*3 +1] eql .(.PATH +2)
01410                                     then ( RETURN .LINEAR[.I*3 +2]);
01420                             if .LINEAR[.I*3] eql EOLIST then (RETURN -1)));
01430                             INDEX =  .INDEX + 1;
01440                             if (.INDEX eql 54) then (FILADDR =.TRACEBUF[ NEXTDESC];
01450                     if .FILADDR eql -1 then RETURN -1;
01460                             DISKIO(2,.FILADDR,TRACEBUF);
01470                             INDEX = 0 );
01480                             STORE = .TRACEBUF[ STORAGE + .INDEX];
01490                     end;
01500              -1
01510     end;
01520
01530
01540     GLOBAL ROUTINE TRACE(ADDR,PPN) =
01550     begin
01560
```

```
01570           % this routines traces the directory containing
01580            the file from the point specified %
01590
01600
01610       while .(.PATH + 4) neq EOLIST do
01620           begin
01630             DISKIO(2,.ADDR,TRACEBUF);
01640              if not PROTCHK(.PPN,3,.TRACEBUF[OWNER],,TRACEBUF[FILSTAT]<protection>)
01650                   then OUTERR(9) ;
01660                 if not .TRACEBUF[ FILSTAT]<kind> then OUTERR(1);
01670             ADDR = SEARCHDIREC(.ADDR);
01680              if .ADDR eql -1 then OUTERR(1);
01690            PATH = .PATH + 4;
01700           end;
01710     DISKIO(2,.ADDR,TRACEBUF);
01720       if not .TRACEBUF[ FILSTAT]<kind> then OUTERR(1);
01730           .ADDR
01740   end;
01750
01760
01770   GLOBAL  ROUTINE FINDCHNL =
01780    begin
01790     Local I;
01800
01810           % this routine looks for a free channel %
01820
01830
01840           I= 0;
01850           While .I lss MAXM do  ( if .ACTIVE[.I,0] eql -1
01860           then RETURN .I  else I = .I + 1  );
01870           -1
01880    end;
01890
01900
01910   GLOBAL  ROUTINE INITCHNL(CHANNEL,FUNCTION,JOB,BUFPTR) =
01920   begin
01930
01940           % this initializes a channel %
01950
01960
01970           ACTIVE[.CHANNEL,JOBNO] = .JOB;
01980           ACTIVE[.CHANNEL,CURBLK] = 0;
01990           ACTIVE[.CHANNEL,SIZE] = .DESCRIPTOR[.CHANNEL,SIZE];
02000           (ACTIVE[.CHANNEL,FILSTAT])<descno> = 0;
02010           ACTIVE[.CHANNEL,FILSTAT]<wrtbit> = .FUNCTION;
02020           ACTIVE[.CHANNEL,FILSTAT]<curmod> = 0;
02030           ACTIVE[.CHANNEL,FILSTAT]<firstmod> = 0;
02040           ACTIVE[.CHANNEL,FILSTAT]<delbits>   = 0;
02050           ACTIVE[.CHANNEL,BUFADDR] = .BUFPTR ;
02060           ACTIVE[.CHANNEL,PARENT] = .DESCRIPTOR[.CHANNEL,CURDESC];
02070   end;
02080
```

02090
02100

```
00010     MODULE PROTCTIONCHECK(NOLIST)=
00020     begin
00030
00040
00050             %  this file contains a routine PROTCHK
00060                to check for validity of access on a file %
00070
00080
00090             %       author      :  AARTI  KUMAR
00100               date of creation:  15 JUNE, 1982      %
00110
00120
00130
00140             MACRO
00150                     protection       = 0,9$, ! file-protection specified
00160                     prog             = 0,8$, ! programmer number of user's ppn
00170                     proj             = 8,8$, ! project number of user's ppn
00180                     prg              = 0,3$, ! protection for all user's
00190                     prj              = 3,3$, ! protection for user's of same project
00200                     onr              = 6,3$, ! protection for owner
00210                     SYSPPN           = #000401$, ! system ppn ( #001,#001)
00220
00230
00240     GLOBAL  ROUTINE PROTCHK(PPN,FUNCTION,OWNR,LOCK) =
00250     begin
00260       Local VALUE,FLAG;
00270
00280             % this routine is called to do a protection check  %
00290
00300
00310             if ( .PPN eql SYSPPN ) then RETURN 1 ;
00320             if .PPN<proj> eql .OWNR<proj>
00330                 then if .PPN<prog> eql .OWNR<prog>
00340                     then VALUE =  .LOCK<onr>
00350                     else VALUE =  .LOCK<prj>
00360                 else     VALUE =  .LOCK<prg> ;
00370             if .VALUE leq .FUNCTION then (RETURN 1 )
00380             else (RETURN 0);
00390     end;
00400
00410
00420
00430     end ELUDOM
```

```
00010
00020
00030          % definitions of all variable declarations in disk manager  %
00040
00050          % author : AARTI KUMAR ;  1 JULY  ,1982   %
00060
00070
00080     external STATUS,LASTJOB;
00090     external REQUEST,BUFFER;
00100
00110     structure ARRAY[PTR,REC] =[num*ind*2](.ARRAY +.PTR*ind*2  +.REC*2)<0,8*2>;
00120
00130     map ARRAY REQUEST;  ! REQUEST TABLE FOR DISK
00140
00150     map VECTOR  BUFFER;
00160
00170     external STARTIO , AWAKE;
00180
00190        MACRO
00200                    IN = STATUS<0,4>$,
00210                   OUT = STATUS<4,4>$,
00220               PENDING = STATUS<8,1>$,
00230                AWAKEN = STATUS<9,1>$,
00240                  FLAG = STATUS<10,1>$,
00250                 JOBNO = 0$,
00260               IOSTAT = 1$,
00270               BLKADD = 2$,
00280                 LOCN = 3$,
00290                 MAXM = 40$,
00300              DISKBLK = REQUEST[.OUT,BLOCK]$,
00310             LOCATION = REQUEST[.OUT,LOCN]$,
00320            operation = 0,1$,
00330            extension = 14,2$,
00340               ERRVAL = #077000$,
00350               CURBLK = 1$,
00360              CURSTAT = 0$,
00370              CURLOCN = 2$,
00380                WRDNO = 3$;
00390
00400
00410
00420
00430
00440     external DSKERR;
00450
00460     MACRO REMOVE(JOB,NUMB)=
00470             ( DSKERR =  (NUMB + ERRVAL) ; RETURN )$;
```

```
00010     MODULE DIVIDE(NOLIST)=
00020     BEGIN
00030
00040      GLOBAL ROUTINE MODL(DIVIDEND,DIVISOR)=
00050     BEGIN
00060      WHILE .DIVIDEND GEQ .DIVISOR DO
00070            BEGIN
00080            DIVIDEND = .DIVIDEND - .DIVISOR;
00090            END;
00100       .DIVIDEND
00110     END;
00120
00130
00140     GLOBAL ROUTINE DIVL(DIVIDEND,DIVISOR) =
00150     BEGIN
00160       LOCAL QUOTIENT;
00170            QUOTIENT = 0;
00180         WHILE .DIVIDEND GEQ .DIVISOR DO
00190            BEGIN
00200               DIVIDEND = .DIVIDEND - .DIVISOR; QUOTIENT = .QUOTIENT + 1;
00210            END;
00220       .QUOTIENT
00230     END;
00240
00250     END ELUDUM
```

```
00010   MODULE DISKMANAGER(NOLIST) =
00020   begin
00030
00040
00050           % DISK MANAGER FOR DISK I/O REQUESTS  %
00060
00070           %       AUTHOR          :  AARTI  KUMAR
00080             DATE OF CREATION : 1 JULY, 1982      %
00090
00100
00110
00120      MACRO
00130                 IN = STATUS<0,4>$,          ! index to next free entry in table
00140                OUT = STATUS<4,4>$,          ! next request to be carried out
00150            PENDING = STATUS<8,1>$,          ! (0/1 - interrupt pending no/yes)
00160             AWAKEN = STATUS<9,1>$,          ! ( 0/1- lastjob to ba awakened yes/no)
00170               FLAG = STATUS<10,1>$,         ! ( descriptor read operation)
00180              JOBNO = 0$,                    ! user's job number field
00190             IOSTAT = 1$,                    ! status word stored with each request
00200             BLKADD = 2$,                    ! block address for disk i/o
00210               LOCN = 3$,                    ! memory location for disk i/o
00220             latter = 15,1$,                 ! position of descriptor in block
00230               MAXM = 40$,                   ! number of requests possible
00240            DISKBLK = REQUEST[.OUT,BLKADD]$, ! points to block address in request
00250           LOCATION = REQUEST[.OUT,LOCN]$,   ! points to memory location
00260             ERRVAL = #077000$,              ! error value range starts
00270          operation = 0,1$,                  !(0/1-read/write operation)
00280          extension = 14,2$,                 ! ( memory extension bits stored
00290           function = 0,3$,                  ! ( operation specified by user)
00300             CURBLK = 1$,                    ! block address of current request
00310            CURSTAT = 0$,                    ! status word of current request
00320            CURLOCN = 2$,                    ! memory location for current request
00330              WRDNO = 3$;          ! two's complement of number of words transfered
00340
00350
00360
00370   STRUCTURE TABLE[NO,FIELD] =
00380           [NO*FIELD*BYTES](.TABLE + .NO * FIELD * BYTES +.FIELD * BYTES )<0,8*BYTES> ;
00390
00400   word global TABLE REQUEST[40,4] ;   ! request table for all disk i/o
00410
00420   word global VECTOR BUFFER[128] ;    ! local buffer used for descriptor operations
00430
00440   external IOTABLE;
00450   map VECTOR IOTABLE;  ! stores current request
00460
00470   global DSKERR;
00480
00490   GLOBAL STATUS ,           ! status word for disk manager
00500          LASTJOB ;          ! stores last job number
00510
00520   external INITIO,SETPRIOR,RESETPRIOR,BLOCK;
```

```
00530
00540     MACRO OUTERR(NUMB)=
00550             ( DSKERR =  (NUMB + ERRVAL) ;  RETURN -1)$;
00560
00570
00580     GLOBAL  ROUTINE STARTIO =
00590      begin
00600        Local PTR ;
00610
00620
00630             % this routine is called to initiate the operation specified
00640               by the user  %
00650
00660
00670             case  .REQUEST[.OUT,IOSTAT]<function>  of
00680                set
00690                      !0 - READ BLOCK
00700                      (AWAKEN = 1 ;
00710                              IOTABLE[CURSTAT]<operation> = 0;
00720                              IOTABLE[CURSTAT]<extension> = .REQUEST[.OUT,IOSTAT]<extension> ;
00730                              IOTABLE[CURBLK]  = .DISKBLK;
00740                              IOTABLE[CURLOCN] = .LOCATION ;
00750                              IOTABLE[WRDNO] = -128 ;
00760                      INITIO()  );
00770
00780
00790                      !1 - WRITE BLOCK
00800                      (AWAKEN = 1 ;
00810                              IOTABLE[CURSTAT]<operation> = 1;
00820                      IOTABLE[CURSTAT]<extension> = .REQUEST[.OUT,IOSTAT]<extension> ;
00830                              IOTABLE[CURBLK]  = .DISKBLK;
00840                              IOTABLE[CURLOCN] = .LOCATION;
00850                              IOTABLE[WRDNO] = -128;
00860                       INITIO() );
00870
00880
00890                      !2- READ DESCRIPTOR
00900                      (AWAKEN = 0 ; REQUEST[.OUT,IOSTAT] = 4 ;
00910                              IOTABLE[CURSTAT]<operation> = 0;
00920                              IOTABLE[CURBLK] = .DISKBLK;
00930                              IOTABLE[CURLOCN] = BUFFER;
00940                              IOTABLE[WRDNO] = - 128;
00950                      OUT = .OUT - 1;
00960                      INITIO()   );
00970
00980
00990                      !3 - WRITE DESCRIPTOR
01000                      (AWAKEN = 0 ; REQUEST[.OUT,IOSTAT] = 5;
01010                              IOTABLE[CURSTAT]<operation> = 0;
01020                              IOTABLE[CURBLK] = .DISKBLK;
01030                              IOTABLE[CURLOCN] = BUFFER;
01040                              IOTABLE[WRDNO] = - 128;
```

```
01570              RESETPRIOR();
01580              end;
01590      end;
01600
01610
01620      GLOBAL ROUTINE INTSTAT=
01630       begin
01640
01650
01660              % initializes the disk manager  %
01670
01680
01690              Incr I from 0 to 39 do
01700                begin
01710                      Incr J from 0 to 3 do
01720                        begin
01730                          REQUEST[.I,.J] = 0;
01740                         end;
01750                end;
01760              Incr I from 0 to 127 do ( BUFFER[.I] = 0 ) ;
01770              STATUS = 0; LASTJOB = 0;
01780              DSKERR = 0;
01790              Incr I from 0 to 3 do ( IOTABLE[.I] = 0 );
01800        end;
01810
01820
01830
01840
01850
01860
01870      end ELUDOM
```

```
00010     MODULE  DSKDRIVER(NOLIST)=
00020     begin
00030
00040
00050           % DISK  DRIVER   %
00060
00070           %      AUTHOR       :  AARTI  KUMAR
00080            DATE OF CREATION  :  2 JULY, 1982      %
00090
00100
00110
00120       MACRO
00130
00140                 retries  = 0,4$,          ! number of retries stored
00150                     num  = 40$,           ! maximum number of requests
00160                     ind  = 4$,            ! no of fields in each request
00170                  errbit  = 15,1$,         ! error condition indicator
00180                 SECSIZE  =  -128$,        ! 2's complement of sector size
00190
00200           % command bits in the command register %
00210
00220                  sekbit  = 0,1$,          ! (0/1 - seek coomand off/on)
00230                  serbit  = 1,1$,          ! search command specified
00240                  wrtbit  = 3,1$,          ! write command specified
00250                 readbit  = 2,1$,          ! read command specified
00260                   enabl  = 6,1$,          ! interrupt enabled
00270                 headadv  = 7,1$,          ! head advance command
00280                 memextn  = 8,2$,          ! memory extension specified
00290                    unit  = 10,1$,         ! unit of disk drive
00300                    port  = 11,2$,         ! port-( 01)
00310                    varb  = 13,1$,         ! ( no of words variable )
00320                    mask  = 14,1$,         ! interrupt mask bit
00330
00340           % field specifications in disk block %
00350
00360                     trk  = 0,8$,          ! track  field in drive register
00370                   sectr  = 0,5$,          ! sector field in drive register
00380                   surfc  = 5,4$,          ! surface field in drive register
00390
00400           % bits in control and status register %
00410
00420                  unsafe  = 8,1$,          ! drive unsafe
00430                    done  = 14,1$,         ! operation done
00440                conready  = 7,1$,          ! controller ready
00450                 crccerr  = 2,1$,          ! crcc error in write
00460                    busy  = 0,1$,          ! controller busy
00470                  nonmem  = 12,1$,         ! error indicates wrong memory
00480
00490           % bits in attention register %
00500
00510                   port1  = 3,1$,          ! port 1 specified
00520                   port2  = 2,1$,          ! port2 specified
```

```
00530                   port3   = 1,1$,         ! port3 specified
00540                   port4   = 0,1$,         ! port4 specified
00550
00560           % bits in drive status register %
00570
00580                  online   = 5,1$,         ! drive on -line
00590                   ready   = 6,1$,         ! drive ready
00600                  reject   = 4,1$,         ! operation rejected
00610                  addchk   = 3,1$,         ! address check
00620                  sekcom   = 1,1$,         ! seek complete if set
00630                  sercom   = 0,1$,         ! search complete if set
00640                sekincom   = 2,1$,         ! seek incomplete
00650                  wrtprot  = 15,1$,        ! port write protected if set
00660
00670           % mapping of block address to track, sector, surface %
00680
00690                cylinder   = 7,8$,         ! cylinder specified in logical address
00700                  filblk   = 0,7$,         ! file block number ( logical)
00710                  latter   = 15,1$;        ! position specified of descriptor
00720
00730
00740       BIND
00750                   BLKREG  = #177444,      ! sector & surface register
00760                  ATTNREG  = #177446,      ! attention register
00770                  DRVSTAT  = #177450,      ! drive status register
00780                   TRKREG  = #177452,      ! track register
00790                   COMREG  = #177454,      ! command register
00800                CNTRLSTAT  = #177456,      ! controller status register
00810                   ADDREG  = #177460,      ! address register
00820                   WRDCNT  = #177462,      ! word count register
00830                   WRTCHK  = #004040,      ! operation - write check
00840                    SENSE  = #104000,      ! operation ( sense)
00850                  RESTORE  = #004020,      ! operation - restore head to trk 0
00860                   TRPVEC  = #10,          ! trap vector location
00870                     DTVT  = #200,         ! data interrupt vector
00880                    NDTVT  = #204;         ! non-data interrupt vector
00890
00900
00910   MACRO READCOMMAND=
00920     begin               ! read command
00930           TEMP = 0;
00940           TEMP<unit> = 0;
00950           TEMP<port> = 1;
00960           TEMP<enabl> = 1;
00970           TEMP<sekbit> = 1;
00980           TEMP<serbit> = 1;
00990           TEMP<readbit> = 1;
01000     end$;
01010
01020
01030   MACRO WRTCOMMAND=
          begin             ! write command
```

```
01050              TEMP = 0;
01060              TEMP<unit> = 0;
01070              TEMP<port> = 1;
01080              TEMP<enabl> = 1;
01090              TEMP<sekbit> = 1;
01100              TEMP<serbit> = 1;
01110              TEMP<wrtbit> = 1;
01120        end$;
01130
01140     global TEMP,ERROR;
01150
01160     word global VECTOR IOTABLE[4];
01170
01180     require DISK1.B11;
01190
01200     external DIVL,MODL;
01210
01220
01230     GLOBAL ROUTINE INITIO=
01240        begin
01250
01260
01270              % this routine initiates a disk read/write
01280               by entering in all controller registers  appropriate values  %
01290
01300
01310               TEMP=0;  ERROR<errbit> =0 ;
01320              until .CNTRLSTAT<conready> eql 1 do ;
01330                   DO (COMREG = SENSE ) Until .DRVSTAT<ready> eql 1  ;
01340              TRKREG<trk> = .IOTABLE[CURBLK]<cylinder>;
01350              BLKREG<sectr> = MODL( .IOTABLE[CURBLK]<filblk> , 10) + 1;
01360              BLKREG<surfc> = DIVL( .IOTABLE[CURBLK]<filblk> , 10 );
01370              WRDCNT = .IOTABLE[WRDNO];
01380              ADDREG = .IOTABLE[CURLOCN];
01390              case .IOTABLE[CURSTAT]<operation> of
01400                   set
01410                        !0= read
01420                        ( READCOMMAND;
01430                          TEMP<memextn> = .IOTABLE[CURSTAT]<extension> ;
01440                        if .IOTABLE[WRDNO] lss SECSIZE then TEMP<varb> = 1;
01450                        COMREG = .TEMP );
01460
01470                        !1=write
01480                        ( WRTCOMMAND;
01490                          TEMP<memextn> = .IOTABLE[CURSTAT]<extension> ;
01500                        if .IOTABLE[WRDNO] lss SECSIZE then TEMP<varb> = 1 ;
01510                        COMREG = .TEMP );
01520                   tes;
01530        end;
01540
01550
```

```
01570
01580
01590
01600     ROUTINE INTERRUPT DATATRANSFER=
01610       begin
01620
01630             % interrupt servicing for data transfer interrupts  %
01640
01650
01660             if .ATTNREG<port2> then
01670                 begin
01680                   COMREG = SENSE;
01690                   if not .DRVSTAT<ready> then
01700                     begin
01710                         COMREG = RESTORE;
01720                         ERROR<retries> = .ERROR<retries> + 1;
01730                         if .ERROR<retries> eql 5 then
01740                             ( ERROR<errbit> = 1 ; ERROR<RETRIES> =0 ) else INITIO();
01750                     end;
01760                 end
01770       else   begin
01780                   if .CNTRLSTAT<errbit> then
01790                     begin
01800                         if .CNTRLSTAT<nonmem> or .CNTRLSTAT<unsafe> then
01810                                 ( ERROR<errbit> = 1)
01820                         else begin
01830                                 COMREG = RESTORE;
01840                                 ERROR<retries> = .ERROR<retries> + 1;
01850                                 if .ERROR<retries> eql 5 then ( ERROR<errbit> = 1  ;
01860                                         ERROR<RETRIES> =0 ) else INITIO();
01870                                 end;
01880                     end;
01890               end;
01900             if .ERROR<errbit> then REMOVE(.LASTJOB,13);
01910             do begin
01920                   PENDING = 0; FLAG = 0;
01930                   if .AWAKEN then AWAKE(.LASTJOB) ;
01940                   if (.IN neq .OUT) or ( not .AWAKEN)
01950                     then begin
01960                           LASTJOB = .REQUEST[.OUT,JOBNO] ;
01970                           STARTIO();
01980                           PENDING = 1;
01990                           OUT = .OUT + 1;
02000                           if .OUT eql MAXM then OUT = 0;
02010                         end
02020                     else FLAG = 0;
02030               end
02040             until not .FLAG;
02050       end;
02060
02070
            ROUTINE INTERRUPT NONDATA=
```

```
02090       begin
02100
02110             % interrupt servicing for non data commands  %
02120
02130
02140             if .ATINREG<port2> then
02150                 begin
02160                   COMREG = SENSF;
02170                   if not .DRVSTAT<ready> then
02180                       begin
02190                           COMREG = RESTORF;
02200                           ERROR<retries> = .ERROR<retries> + 1;
02210                           if .ERROR<retries> eql 5 then
02220                             ( ERROR<errbit> = 1 ; ERROR<RETRIES> =0 ) else INITTO();
02230                       end  ELSE ERROR<errbit> =0;
02240                 end;
02250       end;
02260
02270
02280
02290       GLOBAL ROUTINE INITDISK=
02300         begin
02310
02320
02330             % initializes disk controller  %
02340
02350
02360             DTVT = DATATRANSFER;
02370             (DTVT + 2) = #340;
02380             NDTVT = NONDATA;
02390             (NDTVT + 2) = #340;
02400             DO ( COMREG = #104000) UNTIL .DRVSTAT<ONLINE> EQL 1;
02410
02420         end;
02430
02440
02450
02460       end  ELUDOM
```

```
00010    MODULE  FILLBUFFERS( NOLIST) =
00020     begin
00030
00040           % LINE PRINTER BUFFER MANAGER %
00050
00060        %        AUTHOR      : AARTI KUMAR
00070          DATE OF CREATION : JULY 4, 1982  %
00080
00090
00100
00110       MACRO
00120            EOLN     =    #12$,
00130            RNGSIZE  =     15$,                    ! no of buffers available
00140            TYPE     =     0$,                     ! type of buffer field
00150            LINK     =     1$,                     ! ptr to next of similar type
00160            EM       =     0$,                     ! empty (buffer status)
00170            IN       =     1$,                     ! INput (buffer status)
00180            OUT      =     2$,                     ! output (buffer status)
00190            waiton   =    LPTSTAT<10,1>$,          ! bit indicates file block wait
00200            pending  =    LPTSTAT<11,1>$,          ! bit indicates interrupt pending
00210            intron   =    LPTSTAT<12,1>$,          ! bit indicates interrupt servicing is on
00220            BLANK    =   #40$,                     ! blank character
00230            MSG1     =    -1$,                     ! wait for next file block
00240            CEM      =    LPTSTAT<0,5>$,           ! count of no of empty buffers
00250            CIN      =    LPTSTAT<5,5>$;           ! count of no of INput full buffers
00260
00270       global LPTSTAT ,     ! current   status of all buffers and driver
00280              INPTR ,     ! pointer to next character in current buffer
00290               BANRPTR ;        ! ptr to buffer containing banner
00300
00310       structure RING[no,cnt] = [no*cnt*bytes](.RING + .no*cnt*bytes  + .cnt*bytes)<0,8*bytes>;
00320       structure LNKRNG[h,k] = [h*k*bytes]( .LNKRNG + .h*k*bytes + .k*bytes)<0,8*bytes>;
00330
00340       byte global RING LPTBUF[15,134];          ! ring of one line LPTBUFs
00350       byte global LNKRNG BS[15,2];              ! current status of each buffer
00360       byte global VECTOR F[2] , ! ptr to first of each type
00370                 VECTOR CUR[4] ; ! ptr to current of each type
00380
00390       external FILEBUF ;          ! file buffer for currently opened file
00400       map VECTOR FILEBUF;
00410
00420       external IDENT;
00430       map VECTOR IDENT;         ! stores ascii form of banner
00440
00450       external CURPTR ;           ! ptr to next character in file buffer
00460       external PUTLINE;
00470       external PSEM,VSEM;                  ! counting semaphores
00480       external SETPRIOR,RESETPRIOR;
00490
```

```
00530     GLOBAL routine BUFINIT=
00540       begin
00550
00560         % this is an initializing procedure for the driver %
00570
00580        Local I;
00590
00600             LPTSTAT = 0;
00610             BANRPTR = -1;
00620             INPTR = 0;
00630             CEM   = RNGSIZE;
00640             CIN   = 0 ;
00650             F[EM] = RNGSIZE -1;
00660             F[IN] = -1;
00670             CUR[EM] = 0;
00680             CUR[IN] = -1;
00690             CUR[OUT] = -1;
00700             I = RNGSIZE -1;
00710              do begin
00720                   BS[.I,LINK] = .I-1;
00730                   I = .I -1;
00740                 end
00750              until (.I eql 0);
00760             BS[.I,LINK] = -1;
00770              INCR J from 0 to (RNGSIZE-1) do
00780                     (BS[.J,TYPE] = EM);
00790                       pending = 0;
00800             waiton = 0; intron = 0;
00810             Incr J from 0 to 14 do
00820               begin
00830                     Incr L from 0 to 133 do
00840                       begin
00850                         LPTBUF[.J , .L] = 0 ;
00860                       end;
00870               end;
00880
00890
00900       end;
00910
00920
00930     global routine TAKEBUF(KIND)=            ! get next buffer of type "KIND"
00940       begin
00950         Local TEMP,VALUE;
00960
00970             VALUE = .F[.KIND];
00980             TEMP =  .BS[ .F[.KIND] , LINK];
00990             F[.KIND] = .TEMP;
01000          .VALUE
```

```
01050         Local TEMP;
01060
01070              TEMP = .CUR[.KIND];
01080              BS[.TEMP,LINK] = .NUM;
01090              BS[.NUM,TYPE] = .KIND;
01100              BS[.NUM,LINK] = -1;
01110              CUR[.KIND] = .NUM;
01120       end;
01130
01140
01150       GLOBAL routine SENDLINE=
01160         begin
01170
01180          % this is a procedure for sending a line at a time
01190               to one of a ring of buffers %
01200
01210               SETPRIOR();
01220               If not .waiton then
01230                  begin
01240                       INPTR = 0;
01250                       PSEM( EM,10);
01260                       CUR[IN] = TAKEBUF(EM);
01270                  end;
01280               while (.FILEBUF[.CURPTR] neq EOLN)
01290                 do begin
01300                       LPTBUF[.CUR[IN] , .INPTR] = .FILEBUF[.CURPTR];
01310                       INPTR = .INPTR +1;
01320                       CURPTR = .CURPTR + 1;
01330                    If ( .CURPTR eql 256 ) then ( waiton = 1; RETURN MSG1);
01340                                  ! block ( wait for disk i/o )
01350                    end;
01360               LPTBUF[ .CUR[IN] , .INPTR] = EOLN;
01370               waiton = 0;
01380               ADDBUF( .CUR[IN] , IN);
01390               VSEM( IN,11);
01400               If not .pending then ( intron = 0;  PUTLINE()  );
01410               RESETPRIOR();
01420         1
01430       end;
01440
01450
01460
01470       GLOBAL routine BANNER=
01480         begin
01490        Local PTR;
01500
01510
01520               % printing of banners %
```

```
01570              BANRPTR = .CUR[IN];
01580              Incr I from 0 to 5 do (LPTBUF[.CUR[IN] ,.I] = BLANK );
01590                    INPTR = 6;  PTR = 0;
01600              While ( .PTR lss 7 ) do
01610                    ( LPTBUF[.CUR[IN] , .INPTR] = .IDENT[.PTR] ;
01620                                 INPTR = .INPTR + 1; PTR = .PTR + 1);
01630              LPTBUF[.CUR[IN],.INPTR] = EOLN;
01640              ADDBUF(.CUR[IN] , IN);
01650              VSEM( IN,11);
01660              If not .pending then (intron = 0 ; PUTLINE() );
01670              RESETPRTOR();
01680        end;
01690
01700
01710
01720
01730
01740
01750     end ELUDOM
```

```
00010     MODULE LPTSPOOLER(NOLIST)=
00020      begin
00030
00040
00050       % LINE PRINTER SPOOLER PROGRAM %
00060
00070       %       AUTHOR       :  AARTI KUMAR
00080        DATE OF CREATION :  JULY 2, 1982 %
00090
00100
00110
00120       % THIS FILE CONTAINS TWO MAIN ROUTINES
00130         (1) ENTERSPOOL :this routine enters print requests
00140             in the queue maintained by the spooler
00150         (2) LPTSPL: this routine runs as a user process
00160             which accesses the next request in the queue
00170             (for printing)  and opens and transfers
00180             the file to the printer    %
00190
00200       MACRO
00210             MAXSEQ    = 1000$,    ! MAX-limit of sequence no
00220             INITSEQ   = 100$,     ! min-limit of sequence no
00230             SEQNO     = 0$,       ! field specification for seq-no's in spool table
00240             JOBNO     = 1$,       !  field specification for job-no's in spool table
00250             FNAME     = 2$,       ! pointer to filename
00260             BANR      = 6$,       !field specification for user's ppn
00270             PAGE      = 7$,       ! field specification for page-limit
00280             SIN       = POINTER<0,6>$,      !pointer to next free entry in table
00290             SOUT      = POINTER<6,6>$,      ! pointer to entry for next print request
00300             PRNTON    = POINTER<15,1>$,     ! (0/1-  FILE PRINTING OFF/ON )
00310             EOLIST    = '$'$,               ! end of list marker
00320             MSG1      =  -1$,               ! wait for next file block
00330             MAXM      = 50$,          !size of spooler table
00340             SPLJOB    = 50$,        ! job number of spooler
00350             EOF       = -1$;        ! end of file reached
00360
00370
00380
00390        % format of file name : [direc-addr][filename][filename][eolist](4 words) %
00400             % mode of access of requests : [index, <field-specification>] %
00410
00420
00430     structure TABLE[NO,FIELD] = [NO*FIELD*BYTES](.TABLE + .NO*FIELD*BYTES + .FIELD*BYTES)<0,8*BY
00440
00450     word global TABLE SPOOL[50,8];   ! table to maintain a queue of all print requests
00460     byte global VECTOR FILEBUF[256]; !buffer to transfer  blocks of open file
00470
00480     global  CURPTR,           ! pointer to next character in buffer
00490             POINTER,          ! status word to store IN and OUT pointers of table
                                    ! number returned by OPEN
```

```
00530
00540
00550     external SENDLINE;     ! routine to send a line to a ring of one line buffers
00560     external BANNER;       ! routine to print banners for a file
00570     external ABORT  ;  !routine to abort jobs
00580     external OPEN,GET,CLOSE;       ! routines to open and read file blocks
00590     external BLOCK,AWAKE,SETPRIOR,RESETPRIOR;
00600
00610     routine SEQ=
00620       begin
00630             LASTSEQ = .LASTSEQ + 1;
00640             If ( .LASTSEQ eql MAXSEQ ) then (LASTSEQ = INITSEQ);
00650         .LASTSEQ
00660       end;
00670
00680
00690     byte global  VECTOR IDENT[8];
00700
00710     routine CNVRTOCT=
00720       begin
00730
00740             IDENT[0] = .SPOOL[.SOUT,BANR]<14,2>  + #60 ;
00750             IDENT[1] = .SPOOL[.SOUT,BANR]<11,3>  + #60 ;
00760             IDENT[2] = .SPOOL[.SOUT,BANR]<8,3> + #60 ;
00770             IDENT[3] = #54;
00780             IDENT[4] = .SPOOL[.SOUT,BANR]<6,2>  + #60 ;
00790             IDENT[5] = .SPOOL[.SOUT,BANR]<3,3>  + #60 ;
00800             IDENT[6] = .SPOOL[.SOUT,BANR]<0,3>  + #60 ;
00810       end;
00820
00830
00840     global routine INITSPOOL=
00850       begin                    ! this returns the next sequence number available
00860             LASTSEQ = 99;
00870             SIN = 0; SOUT = 0;
00880             POINTER = 0;
00890             PRNTON = 0;
00900             Incr I from 0 to 49 do
00910                begin
00920                     Incr J from 0 to 7 do
00930                       begin
00940                          SPOOL[.I , .J] = 0 ;
00950                       end;
00960                end;
00970
00980       end;
00990
01000
```

```
01050       % this routine enters requests in the spooler table %
01060
01070
01080             SETPRIOR();
01090         If ( ( .SIN eql .SOUT ) and .PRNTON ) then ABORT(.JOB,40); ! table full
01100             SPOOL[.SIN,SEQNO ]   = SEQ();
01110             SPOOL[.SIN,JOBNO ]   = .JOB;
01120             SPOOL[.SIN,FNAME ]   = .(.PATH);
01130             SPOOL[.SIN,FNAME+1]  = .(.PATH + 2);
01140             SPOOL[.SIN,FNAME+2]  = .(.PATH + 4);
01150             SPOOL[.SIN,FNAME+3]  = EOLIST;
01160             SPOOL[.SIN,BANR ]    = .PPN;
01170             SPOOL[.SIN,PAGE ]    = .LIM;
01180          SIN =  .SIN + 1;
01190             if ( .SIN eql MAXM ) then (SIN = 0);
01200             if not .PRNTON then AWAKE();
01210             RESETPRIOR();
01220       end;
01230
01240
01250     routine TRANSFER=
01260      begin
01270
01280       % this routine transfers a single file block  to a ring
01290          of buffers maintained by the spooler for printing lines %
01300
01310       Local VALUE;
01320
01330        CURPTR = 0;
01340        While ( .CURPTR lss 256 ) do
01350        begin
01360             VALUE = SENDLINE();
01370             If ( .value eql MSG1 ) then RETURN ;
01380             CURPTR = .CURPTR + 2;
01390             if ( .FILEBUF[.CURPTR] eql #14   ) then RETURN ;
01400        end;
01410      end;
01420
01430
01440       OWN VAL2;
01450     WORD OWN VECTOR VAL[4] ;
01460
01470
01480     ROUTINE NEXTOUT=
01490      begin
01500
01510             % this routine takes the next request for printing
01520                    from the queue %
```

```
01570             INCR I FROM 0 TO 3 DO ( VAL[.I] = .SPOOL[.SOUT,FNAME + .I] ) ;
01580             VAL2 = .SPOOL[.SOUT,BANR] ;
01590             CNVRTOCT();
01600             RESETPRIOR();
01610       end;
01620
01630
01640       ROUTINE INCROUT=
01650         begin
01660
01670             % this increments the out pointer %
01680
01690             SETPRIOR();
01700             PRNTON = 0;
01710             SOUT = .SOUT + 1;
01720             if ( .SOUT eql MAXM   )  then ( SOUT = 0 ) ;
01730             RESETPRIOR();
01740       end;
01750
01760
01770       GLOBAL ROUTINE LPTSPL=
01780        begin
01790
01800          % this routines runs as a user process and repeatedly opens
01810                and transfers the next file in the table for printing %
01820
01830
01840           while   1  do
01850             begin
01860                   NEXTOUT();
01870               CHANL = OPEN(SPLJOB,.VAL2,VAL,0,FILEBUF);
01880               BANNER();
01890               do begin
01900                        FILBLK = GET(SPLJOB,.CHANL);
01910                        TRANSFER();
01920                  end
01930               until  .FILBLK eql EOF;
01940               CLOSE(SPLJOB,.CHANL);
01950               INCROUT();
01960               end;
01970          end;
01980
01990
02000
02010
02020
02030
02040
02050       end ELUDOM
```

```
00010   MODULE LINEPRINTER(NOLIST)=
00020    begin
00030
00040           %     LINE PRINTER DRIVER     %
00050
00060           %          AUTHOR          :   AARTI KUMAR
00070             DATE OF CREATION   :  19 JULY, 82    %
00080
00090
00100     MACRO
00110           prntcom       = 0,1$,       ! bit to set print command
00120           eject         = 1,1$,       ! bit to give command to eject paper
00130           errmask       = 5,1$,       ! mask for errors
00140           enabl         = 6,1$,       ! allow enabling of interrupts
00150           done          = 7,1$,       ! set when printer can accept new character
00160           ready         = 8,1$,       ! set when printer can accept new line
00170           buffull       = 10,1$,      ! indicates printer buffer full when set
00180           redymask      = 9,1$,       ! mask for ready interrupt
00190           xion          = 11,1$,      ! recover from error
00200           parity        = 14,1$,      ! parity error in printer
00210           data          = 0,6$,       ! bits to store a character fed into buffer
00220           vfuline       = 6,1$,       ! bit to specify that VFU character is present
00230           imdt          = 5,1$,       ! paper movement before/after controlled
00240           vfucom        = 0,5$,       !specifies no of lines to move
00250           charcnt       = 0,8$,       ! specifies no of characters entered per line
00260           lincnt        = 8,8$,       ! specifies lines printed from channel 1
00270           EM            = 0$,         ! empty buffer status
00280           IN            = 1$,         ! full buffer status
00290           OUT           = 2$,         ! indicates state of buffer
00300           EOLN          = #12$,       ! line feed character
00310           BLANK         = #40$,       ! (blank character)
00320           banron        = 1$,         ! banner printing is on
00330           pjeject       = 0$,         ! page eject is over
00340           dummy         = 2$,         ! dummy line is to be printed
00350           line          = 3$,         ! new line is to be printed
00360           pending       = LPTSTAT<11,1>$,    ! interrupt pending;
00370           intron        = LPTSTAT<12,1>$,  ! interrupt servicing is on
00380           CEM           = LPTSTAT<0,5>$, ! count of empty buffers
00390           CIN           = LPTSTAT<5,5>$, ! count of full buffers
00400           CR            = #15$;       ! carriage return
00410
00420
00430
00440     BIND
00450           LCSR      = #777630,        ! line printer control and command register
00460           LPBR      = #777632,        ! line printer data register
00470           LVFR      = #777634,        ! line printer VFU register
00480           CCSR      = #777636,        ! line printer character and line count register
00490           ERRINTR     = #160,         ! error interrupt vector
00500           REDYINTR    = #164;         ! ready interrupt vector
00510
00520
```

```
00530      GLOBAL CURCOND,LCOMND ;
00540
00550
00560      external  TAKEBUF, ADDBUF ;
00570      external LPTSTAT,BANRPTR;
00580      external LPTBUF, CUR;
00590      external PSEM ,VSEM ;
00600
00610      structure ARY[no,ind] = [15*134*BYTES](.ARY + .no*134*BYTES + .ind*BYTES)<0,8*BYTES>;
00620
00630      map ARY LPTBUF;
00640      map VECTOR CUR;
00650
00660
00670      GLOBAL ROUTINE PRNTLINE=
00680       begin
00690
00700            % this routine intiates the printing of a line %
00710
00720        Local TMP1,TMP2;
00730
00740            TMP1 = 0; TMP2 = .LPTBUF[ .CUR[OUT] , .TMP1];
00750            while ( .TMP2 neq EOLN ) do
00760            begin
00770              until .LCSR<done> do ;
00780              if .TMP2<vfuline> then ( TMP2<imdt> = 0 );
00790                   LPBR<data> = .TMP2<data> ;
00800                TMP1 = .TMP1 + 1;
00810               TMP2 = .LPTBUF[ .CUR[OUT] , .TMP1] ;
00820            end;
00830      end;
00840
00850     GLOBAL ROUTINE  PRNTBANR=
00860      begin
00870      Local TMP1 , TMP2;
00880
00890         %  intiates printing of banner  %
00900
00910            TMP1 = 0; TMP2 = .LPTBUF[ .CUR[OUT] ,.TMP1];
00920            incr I from 0 to 9 do
00930                   begin
00940                      until .LCSR<done> do;
00950                      LPBR<data> = BLANK;
00960                   end;
00970            while ( .TMP2 neq EOLN ) do
00980              begin
00990                   until .LCSR<done> do ;
01000                   if .TMP2<vfuline> then ( TMP2<imdt> = 0 );
01010                   LPBR<data> = .TMP2<data> ;
01020                   TMP1 = .TMP1 + 1;
01030                   TMP2 = .LPTBUF[ .CUR[OUT] , .TMP1] ;
01040              end;
```

```
01050      end;
01060
01070
01080      GLOBAL ROUTINE POSTBANR=
01090       begin
01100
01110          %  prints 10 blank lines after banner %
01120
01130               until .LCSR<ready> do ;
01140               until .LCSR<done> do ;
01150                       BANRPTR = -1;
01160                       LVFR<vfuline> = 1;
01170                       LVFR<imdt> = 1;
01180                       LVFR<vfucom> = 10;
01190                       CURCOND = dummy;
01200                       LCOMND = 0 ;
01210                       LCOMND<prntcom> = 1;
01220                       LCOMND<enabl>   = 1;
01230                       LCSR = .LCOMND;
01240        end;
01250
01260      GLOBAL ROUTINE PRNTDUMMY=
01270       begin
01280
01290          % prints a dummy line to shift control to next line %
01300
01310               until .LCSR<ready> do;
01320               until .LCSR<done> do ;
01330               LCOMND = 0;
01340               CURCOND = line;
01350               LCOMND<prntcom> = 1;
01360               LCOMND<enabl> = 1;
01370               LCSR = .LCOMND;
01380       end;
01390
01400
01410      GLOBAL ROUTINE POSTEJECT=
01420       begin
01430
01440          %   leaves a margin of 5 lines after paper eject %
01450
01460               until .LCSR<ready> do;
01470               until .LCSR<done> do;
01480                       LVFR<vfuline> = 1;
01490                       LVFR<imdt> = 1;
01500                       LVFR<vfucom> = 5;
01510               if ( .CUR[OUT] eql .BANRPTR ) then
01520                                       ( PRNTBANR();
01530                                         CURCOND = banron )
01540                        else  ( PRNTLINE() ; CURCOND = dummy );
01550               LCOMND = 0;
01560
```

```
01570              LCOMND<enabl> = 1;
01580              LCOMND<prntcom> = 1;
01590              LCSR = .LCOMND;
01600       end;
01610
01620
01630       GLOBAL ROUTINE PAGEJECT=
01640        begin
01650              until .LCSR<ready> do;
01660              until .LCSR<done> do;
01670                     LCOMND = 0;
01680                     LCOMND<enabl> = 1;
01690                     LCOMND<eject> = 1;
01700                     CURCOND = pieject;
01710                     LCSR = .LCOMND;
01720       end;
01730
01740
01750
01760
01770
01780       GLOBAL ROUTINE PUTLINE=
01790        begin
01800
01810              % intiates printing %
01820
01830              if not PSEM(IN,11)  then RETURN ;
01840              CUR[OUT] = TAKEBUF(IN);
01850              if ( (.CUR[OUT] eql .BANRPTR ) or
01860                           ( .CCSR<lincnt> eql 31) ) then
01870                                  begin  PAGEJECT();
01880                                         pending = 1;
01890                                  end
01900              else begin
01910                           until .LCSR<ready> do ;
01920                            until .LCSR<done> do;
01930                     PRNTLINE();
01940                     LCOMND = 0;
01950                     LCOMND<prntcom> = 1;
01960                     LCOMND<enabl>  = 1;
01970                     pending = 1;
01980                     CURCOND = line;
01990                     LCSR = .LCOMND;
02000                end;
02010       end;
02020
02030
02040       ROUTINE INTERRUPT PRINTER=
02050        begin
02060
```

```
02090           case  .CURCOND of
02100             set
02110                 !0 - pieject
02120                 (  POSTEJECT() );
02130                 !1 - banron
02140                 (  POSTBANP()  );
02150
02160                 !2 - dummy
02170                 ( PRNTDUMMY()  );
02180
02190                 !3 - line
02200                 ( pending = 0;
02210                       ADDBUF( .CUR[OUT] , EM);
02220                       VSEM(EM ,10);
02230                   intron = 1 ; PUTLINE()   );
02240
02250             tes;
02260      end;
02270
02280
02290     ROUTINE INTERRUPT ERRPRNT=
02300      begin
02310
02320      end;
02330
02340
02350
02360     GLOBAL ROUTINE INITPRNT=
02370      begin
02380
02390        %  intialize the printer %
02400
02410             REDYINTR = PRINTER;
02420             ( REDYINTR + 2) = #340;
02430             ERRINTR = ERRPRNT;
02440             (ERRINTR + 2 ) = #340;
02450      end;
02460
02470
02480
02490
02500     end ELUDOM
```

```
00010     MODULE COUNTINGSEMAPHORES(NOLIST)=
00020      begin
00030
00040             % COUNTING SEMAPHORE IMPLEMENTATION %
00050
00060             %       AUTHOR      :  AARTI KUMAR
00070              DATE OF CREATION :  19 JULY,82 %
00080
00090
00100
00110        MACRO
00120             CEM      = LPTSTAT<0,5>$,    !  count of empty buffers
00130             CIN      = LPTSTAT<5,5>$,    !  count of full buffers
00140             intron = LPTSTAT<12,1>$;  ! interrupt servicing is on/off
00150
00160      external LPTSTAT;
00170      external AWAIT,SIGNL;      ! synchronizing routines
00180
00190
00200
00210     GLOBAL routine PSEM(KIND,EVENT)=
00220      begin
00230       Local MUTEX;
00240
00250             % this routine decrements the semaPSEMore
00260               and does a wait if value is less then 0 %
00270
00280
00290             case .KIND of
00300                set
00310                     !0= empty
00320                     ( MUTEX = .CEM );
00330                     !1= full
00340                     ( MUTEX = .CIN );
00350                tes;
00360             If ( .MUTEX eql 0 ) then
00370                     ( if not .intron then ( AWAIT(.EVENT) )
00380                                      else  ( RETURN 0))
00390                     else ( MUTEX = .MUTEX - 1 ) ;
00400                     case .KIND of
00410                       set
00420                          !0= empty
00430                          ( CEM = .MUTEX );
00440                          !1=full
00450                          ( CIN = .MUTEX );
00460                       tes;
00470        1
00480      end;
00490
00500
```

```
00530      begin
00540
00550              % this routine increments the semaPSEMore
00560               and signals the job waiting on this event %
00570
00580              case .KIND of
00590                 set
00600                       !0= empty
00610                       ( CEM = .CEM + 1 );
00620                       !1= full
00630                       ( CIN = .CIN + 1 ) ;
00640                 tes;
00650              SIGNL(.EVENT);
00660      end;
00670
00680
00690
00700
00710      end ELUDOM
```